VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SYSTÉM PRO AKTUALIZACI ANOTACÍ V KORPUSECH

SYSTEM FOR UPDATE OF ANNOTATIONS IN CORPORA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **ŠTĚPÁN VRŠA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JAROSLAV DYTRYCH**
SUPERVISOR

BRNO 2017

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Vrša Štěpán**
Obor: Informační technologie
Téma: **Systém pro aktualizaci anotací v korpusech**
System for Update of Annotations in Corpora
Kategorie: Algoritmy a datové struktury

Pokyny:

1. Seznamte se s formáty velkých korpusových dat využívanými ve Výzkumné skupině znalostních technologií FIT VUT v Brně.
2. Prostudujte nástroje pro zpracování korpusových dat a obvyklé postupy zpracování vedoucí k indexaci korpusu.
3. Navrhněte systém pro opravy a aktualizace anotací v korpusových datech, který umožní zpřístupnění vybraných částí indexovaného korpusu, vizualizaci a editaci stávajících anotací v dostupném anotačním nástroji a přípravu dat pro aktualizaci korpusu v indexovatelném formátu.
4. Implementujte navržené řešení.
5. Zhodnoťte dosažené výsledky a navrhněte možná rozšíření do budoucna.

Literatura:

- Dle doporučení vedoucího.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Dytrych Jaroslav, Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Cílem této práce je vytvoření systému, který umožní uživatelsky přívětivým způsobem zobrazit a aktualizovat anotace velkých korpusových dat ve formátu MG4J. Tato práce analyzuje stávající řešení správy korpusových dat a anotací a stručně popisuje netriviální nástroje SEC a MG4J, které jsou využity v řešení. Celý systém je založen na distribuci dat a serverové komunikaci. Stěžejním prvkem systému je aktualizace anotací v MG4J a následná aktualizace indexů MG4J. Systém je schopný provést výše zmíněné operace s přijatelnou dobou odezvy. Tato práce se také zabývá aktualizací entit ve znalostní databázi.

Abstract

The goal of this thesis is the creation of a system that allows users to display and update the large corpus data annotations in the MG4J format. This thesis analyzes the current corpus data and annotation management solution and briefly describes the non-trivial SEC and MG4J tools used in the solution. The main element of the system is updating annotations in MG4J and subsequently updating the MG4J indexes. The system is capable of performing the above mentioned operations with an acceptable response time. This thesis also deals with updating entities in a knowledge base.

Klíčová slova

MG4J, MG4J_API, SEC, korpus, 4A, znalostní databáze, vertikální text, indexace, aktualizace anotací.

Keywords

MG4J, MG4J_API, SEC, corpus, 4A, knowledge base, vertical text, indexing, annotation update.

Citace

ŠTĚPÁN, Vrša. *Systém pro aktualizaci anotací v korpusech.* Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Dytrych Jaroslav.

Systém pro aktualizaci anotací v korpusech

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jaroslava Dytrycha. Další informace mi poskytli Bc. Jan Doležal a Dávid Prexta. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Štěpán Vrša
14. května 2017

Poděkování

Především bych rád poděkoval panu Ing. Jaroslavu Dytrychovi, který mě provedl procesem psaní práce velice přátelským způsobem a výrazně mně pomohl v dokončení celé bakalářské práce. Jeho odborné rady byly velice užitečné. Dále by rád poděkoval Bc. Janu Doležalovi, který mně pomohl při tvorbě modulu pro SEC. Také chci poděkovat Dávidu Prextovi, který mně poskytl data pro aktualizaci KB.

Obsah

Kapitola 1

Úvod

Tato práce se zabývá návrhem a implementací systému, který bude poskytovat jednotlivé dokumenty v korpusech pro uživatele v čitelné formě a v reálném čase. Další funkcí systému bude zpětná aktualizace anotací v jednotlivých dokumentech. Je třeba si uvědomit, že korpusy jsou obrovské kolekce dat, které obsahují milióny dokumentů. Jejich velikosti se pohybují mezi stovkami GB až po stovky TB. Proto je pro uživatele manuálně náročné aktualizovat jediný atribut anotace v obrovských datech v uspokojivém čase. Tento systém bude schopný vizualizace a aktualizace anotací v obrovských kolekcích dat během desítek sekund.

Kapitola 2 pojednává o korpusech z teoretického hlediska. Obsahuje popisy formátů dat, které jsou nezbytné pro řešení celé práce.

V kapitole 3 je nejdříve popsán postup vzniku korpusů v případě výzkumné skupiny KNOT. V dalších částech kapitola pojednává o nástrojích, které jsou využity v řešení práce. Na konec je popsán předchozí stav správy anotací.

Návrh celého systému se nachází v kapitole 4. V této kapitole se čtenář dozví, jak byl celý systém navrhnut a jak byl rozdělen do různých komponent systému. Také se zde dočte, jak byla navrhnuta komunikace mezi komponenty systému. V samém závěru kapitoly je popsán návrh skriptu, který bude aktualizovat znalostní databázi.

Kapitola 5 obsahuje popis všech nejdůležitějších technologií, které byly v řešení práce využity.

Následující kapitola 6 pojednává o implementaci všech komponent systému. Také je zde obsažen detailnější popis chování a fungování systému. V závěru kapitoly je popsána implementace skriptu aktualizující znalostní databázi.

V kapitole 7 je čtenář obeznámen s výsledky testování komponent systému. Mimo jiné se zde čtenář dočte o rychlosti služeb systému, výsledcích aktualizace znalostní databáze a o problémech, které nebyly vyřešeny.

Poslední kapitola 8 shrnuje dosažené výsledky a obsahuje návrhy možného vylepšení systému do budoucnosti.

Kapitola 2

Korpusová data

Tato kapitola obsahuje definici korpusu z hlediska oboru informatiky. Dále je stručně popsán vertikální text, který je využíván při zpracování dat v projektu Corpora Processing software[1].

Předposlední podkapitola obsahuje popis a strukturu souborů MG4J. Formát MG4J je důležitý, neboť se budu zabývat aktualizací anotací v tomto formátu.

V poslední podkapitole se budu zabývat formátem znalostní databáze. Pochopení formátu znalostní databáze je opět důležité z hlediska její aktualizace, kterou se v této práci také zabývám.

2.1 Korpus

V současnosti se korpusem rozumí rozsáhlý vnitřně strukturovaný a ucelený soubor textů daného jazyka elektronicky uložený a zpracovávaný. Texty jsou v korpusu strukturovány a organizovány se zřetelem k využití pro určitý cíl, vůči němuž pak je korpus považován za reprezentativní. [13]

Existuje mnoho druhů korpusů, které jsou využity pro různé účely analýzy. Některé druhy, jejich účely a příklady korpusů jsou následující [15]:

- **Obecné** – zaměřují se na reprezentování jazyka jako celku – BNC[2]
- **Historické** – korpusy reprezentující starší období jazyka – ARCHER[3], Helsinki Corpus[4]
- **Regionální** – zabývají se regionálním typem jazyka – WCNZE[5]
- **Anotované** – jedná se o korpusy, u kterých byla provedena lingvistická analýza, typicky klasifikace slov – SUSANNE[6]

Korpusy jsou využity v oborech lexikografie, lingvistiky, sociologie, psychologie, umělé inteligence a dalších [13].

[1] http://knot.fit.vutbr.cz/corpproc/corpproc_cs.html
[2] British National Corpus, http://corpus.byu.edu/bnc/
[3] http://www.projects.alc.manchester.ac.uk/archer/
[4] http://www.helsinki.fi/varieng/CoRD/corpora/HelsinkiCorpus/
[5] The Wellington Corpus of Spoken New Zealand English, http://www.victoria.ac.nz/lals/resources/corpora-default/corpora-wsc
[6] http://www.grsampson.net/SueDoc.html

2.2 Vertikální text

Vertikální text je textový formát ukládaný do souborů s koncovkou „`.vert`“. Každý řádek ve vertikálním textu obsahuje jednu pozici (slovo, oddělovač, číslo nebo strukturní značku XML). Každou pozici lze rozšířit o atributy (lemma, tag apod.) přidáváním dalších sloupců oddělených pomocí tabulátoru. Vertikální text obsahuje značky XML, ale nejedná se o validní dokument ve formátu XML, a to především kvůli chybějící kořenové značce dokumentu. Využívané značky jsou popsány v následující tabulce:

Značka	**Význam**
<doc>	Označuje hranici jednotlivého dokumentu ve vertikálním textu. Je to párová značka.
<head>	Reprezentuje titulek dokumentu. Je to párová značka. Za touto značkou mohou následovat metadata dokumentu (datum vytvoření, autor, klíčová slova apod.).
<p>	Označuje odstavec v dokumentu. Párová značka.
<s>	Označuje větu v dokumentu. Párová značka.
<g/>	Jedná se tzv. lepidlo. Slouží k označení dvou pozic, které nebyly odděleny bílým znakem. Nepárová značka.
<link="URL">	Označuje odkaz. Nachází se ve 2. sloupci – tedy vedle pozice na stejném řádku oddělený tabulátorem. Nepárová značka.
<length=N>	Značka, která určuje, pro kolik předchozích pozic platí značka <link>. Nachází se ve 3. sloupci – tedy za značkou <link> oddělená znakem tabulátoru. Nepárová značka.

Tabulka 2.1: Využívané značky ve vertikálním textu

Znalost vertikálního formátu je důležitá pro vytváření korpusů, neboť vytvoření vertikálního textu je jeden z důležitých procesů při zpracování korpusových dat. Zmínka o jeho využití bude v následující kapitole.

2.3 Formát dat MG4J

Formát MG4J je ve Výzkumné skupině znalostních technologií (KNOT) definovaný jako 27 sloupcový soubor[7]. Na každém řádku se nachází jedno slovo (existují výjimky, jako znaky reprezentující čárku, tečku a závorky, které nelze považovat za slovo, spíše je lze označit jako tokeny). V MG4J se nachází speciální řádky, které nesplňují 27 sloupcový formát. První řádek každého souboru má pouze dva sloupce, kde druhý sloupec reprezentuje název

[7] http://knot.fit.vutbr.cz/corpproc/corpproc_cs.html

souboru. Další speciální řádky označují ID dokumentu, URI, ze kterého dokument pochází (`PAGE`), nový odstavec (`PAR`) a větu (`SEN`). Příklady těchto řádků jsou níže:

- **ID** – %%#DOC 2a38c8ac-854b-51a3-8f05-3688fc5c31e9
- **PAGE** – %%#PAGE Brandreth https://en.wikipedia.org/wiki/Gyles_Brandreth
- **PAR** – %%#PAR 1 wx1
- **SEN** – %%#SEN 1 wx1

Sloupce č. 1 až 13 reprezentují syntaktické informace, jako pozice slova ve větě, slovo (token), základní tvar slova apod. Tyto sloupce kromě samotného slova nemají velký význam pro řešení této bakalářské práce. Naopak sloupce č. 14 až 25 nesou informace o anotaci daného slova. Pokud žádná anotace neexistuje, jsou hodnoty těchto sloupců nastaveny na znak „*0*“.

Sloupec č. 14 reprezentuje ID anotace, sloupec č. 15 značí typ anotace, sloupec č. 16 obsahuje URL anotace a sloupec č. 17 obsahuje vizuální reprezentaci anotace. Sloupce č. 18 až 25 lze označit za dynamické. To znamená, že jejich význam se mění dle typu anotace (hodnota sloupce č. 15). Tyto sloupce jsou znázorněny v příloze A.1.

V MG4J se objevují řádky, které sice mají nenulovou hodnotu ve sloupci č. 14, ale mají ostatní anotační sloupce nulové. Taková anotace se nazývá koreferenční anotací. Koreferenční anotace je využita pouze v případě, kde je více řádků se stejnou anotací za sebou. To znamená, že až poslední řádek v pořadí bude obsahovat plnohodnotnou anotaci s nenulovými hodnotami.

2.4 Znalostní databáze

V případě výzkumné skupiny KNOT[8] je znalostní databáze KB (Knowledge Base) soubor ve formátu TSV (Tab-Separated Values) obsahující entity. Každá entita je na jednom řádku a počet sloupců s atributy je závislý na typu entity. Typ entity je dán druhým sloupcem `TYPE`. Sloupce mohou obsahovat více hodnot, pokud jsou označeny uvozovacím slovem `MULTIPLE_VALUES`. Oddělovačem více hodnot je znak „`|`“.

Znalostní databáze v aktuální verzi obsahuje 15 typů entit, kterými jsou `Person`, `Artist`, `Location`, `Artwork`, `Museum`, `Event`, `Nationality`, `Mythology`, `Family`, `Visual_art_form`, `Visual_art_medium`, `Visual_art_genre`, `Art_period_movement`, `Group` a `Other` [16].

Pro řešení bakalářské práce je KB důležitým prvkem. KB je využita v nástroji NER MG4J_API, který spojuje anotace z MG4J a z KB, pokud se jedná o stejnou entitu. Tento krok umožňuje zobrazení více informací, a to z toho důvodu, že MG4J je omezeno pouze na 9 anotačních informací (sloupců), zatímco KB jich obsahuje podstatně více. Aktuální KB je volně dostupná na serveru athena3[9]. Velikost KB je **1,9 GB** a obsahuje až **5 400 000 entit**.

[8] http://knot.fit.vutbr.cz/
[9] http://athena3.fit.vutbr.cz/kb/KB.all

Kapitola 3

Nástroje určené pro zpracování korpusových dat

Před samotnou aktualizací je třeba získat data a upravit je do určitého formátu, se kterým se bude později pracovat. Proto jsem se v první podkapitole rozhodl popsat procesy zpracování dat a vznik korpusů ve výzkumné skupině KNOT[1].

Druhá část obsahuje popis nástroje SEC, který slouží k sémantickému obohacení textu. Jeho důležitost spočívá ve využití tohoto nástroje v řešení práce.

V dalších třech podkapitolách se zabývám nástrojem MG4J a jeho dvěma hlavními funkcemi, indexace a dotazování. V podkapitole indexace zjednodušeně vysvětluji tento náročný proces. Podkapitola o dotazování obsahuje stručný popis této funkce a její možnosti. Příklady dotazů jsou uvedeny v další podkapitole.

V následující podkapitole se zabývám popisem dotazovacího nástroje, který je využit pro grafické zobrazení anotací dokumentů, editaci anotací a dotazování nad daty MG4J.

Poslední podkapitola obsahuje popis anotačního nástroje 4A, který lze považovat za předchozí stav řešení. Popsal jsem stručně systém 4A a jeho části. Také zde zmiňuji podobné systémy a uvádím hlavní přínosy řešení této práce oproti původnímu systému 4A.

3.1 Zpracování dat

Tvorba vlastního korpusu se dělí na několik po sobě jdoucích procesů. V tomto případě bude popsán postup, který se používá ve výzkumné skupině KNOT. Tento projekt se nazývá Corpora Processing software[2].

Prvním krokem je stažení dumpu z Wikipedie[3], projektu CommonCrawl[4] a webových stránek z RSS (Rich Site Summary[5]) kanálů. Data jsou uložena do formátu WARC (Web archive). WARC je archivační formát, který určuje způsob kombinace více digitálním zdrojů do agregovaného archivního souboru spolu se souvisejícími informacemi [19].

Dalším krokem je převedení dat z formátu WARC do vertikálního textu, který je popsán v podkapitole 2.2. Ve vertikalizátoru dochází k odstranění HTML a dokumentů, které

[1] http://knot.fit.vutbr.cz/
[2] http://knot.fit.vutbr.cz/corppproc/corppproc_cs.html
[3] https://dumps.wikimedia.org/
[4] http://commoncrawl.org/
[5] http://www.rssboard.org/rss-specification

nejsou ve vybraném jazyce (typicky angličtina). Poté dochází k tokenizaci[6]. Výstupem vertikalizátoru jsou vertikální soubory.

Třetím krokem je deduplikace, což je odstranění duplikátních dokumentů a odstavců. Po deduplikaci následuje tzv. tagging. Tagging využívá morfologickou analýzu s následnou desambiguací. Zjednodušeně řečeno dochází k získání gramatických informací pro každé slovo v kontextu [9].

Na tagging navazuje další proces a to parsování. Parsování je proces, při kterém se pro každé slovo ve větě určí syntaktická závislost na jiném slovu ve větě. Poté se takovému slovu přiřadí analytická funkce [18].

Předposledním krokem je využití nástroje SEC pro rozpoznání pojmenovaných entit. Výstupem jsou data ve formátu MG4J, tak jak jsou popsány v 2.3. Posledním krokem je indexace, která umožní dotazování nad daty. Indexace je podrobněji popsána v podkapitole 3.4.

3.2 Semantic Enrichment Component

Semantic Enrichment Component[7] *(SEC) poskytuje služby pro sémantické obohacení textu. Poskytuje např. službu anotace textu annotate, hledání entit get_entities a výpisu všech typů a atributů ve znalostní bázi (KB) get_entity_types_and_attributes.* [10]

Důležitou vlastností nástroje SEC je načtení KB do paměti, což vede k rychlému vyhledávání v KB.

SEC umožňuje přidání modulů pro rozpoznávání pojmenovaných entit (NER – Named-entity recognition). Tyto moduly slouží k získání anotací z různých zdrojů. Dostupné moduly NER jsou následující:

- AIDA[8]
- ALCH_API[9]
- COGITO_API[10]

Důležitou službou nástroje SEC je služba `annotate`. Tato služba vrací anotace pro zadaný dokument. Uživatel může sám zvolit, jaký NER bude použit k získání anotací, a to pomocí parametru `enrichment_engine`. Dále lze zvolit výstupní formát anotace nebo nastavit parametr `disambiguate`, který znamená, že se vybere anotace s největší pravděpodobností významu slova. Celý obecný popis požadavku pro službu anotate je následující:

```
{
    "annotate": {
        "input_text": str,
        "annotation_format": [ str, ... ],
        "disambiguate": int,
        "document_uri": str,
        "types_and_attributes": "all" | { str(type): "all" } |
```

[6] http://www.nltk.org/api/nltk.tokenize.html
[7] http://sec.fit.vutbr.cz/
[8] http://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/aida/
[9] https://www.ibm.com/watson/alchemy-api.html
[10] http://cogitoapi.com/

```
        { str(type): [ str(attribute), ... ] },
        "enrichment_engine": str,
        "enrichment_engine_timeout": int,
        "plaintext": bool
    }
}
```

Odpovědí na tento požadavek je anotace v uživatelem zvoleným formátu. Formáty anotací můžou být SXML, XML, HTML, Text, Index, Index2, RDF nebo NIF. Odpověď pak vypadá následovně:

```
{
    "annotation": str
}
```

Další služby, které SEC podporuje, jsou:

- **annotate_vertical** – využívá se pro anotaci textu ve vertikálním formátu,
- **deverticalize** – devertikalizuje vstupní text ve vertikálním formátu,
- **get_enrichment_engines** – vypíše všechny dostupné nástroje NER,
- **get_entities** – vypíše entity z KB, které mají podobné nebo stejné jméno,
- **get_entity_by_uri** – vyhledá entity pomocí zadaného URI,
- **get_entity_types_and_attributes** - vrací všechny dostupné typy a jejich atributy,
- **get_kb_version** – vrací aktuální verzi KB,
- **get_raw_annotations** – vrací řetězec anotací získaný pomocí zadaného nástroje NER.

3.3 MG4J

Managing Gigabytes for Java je volně dostupný indexovací nástroj pro velké sady dokumentů, který je vyvíjen na univerzitě Università degli Studi di Milano pod licencí GNU Lesser General Public License[11]. MG4J podporuje indexaci a následné dotazování nad zaindexovanými daty [4].

3.4 Indexace v MG4J

Indexace je *proces vyjádření obsahu dokumentu pomocí prvků selekčního jazyka, obvykle s cílem umožnit zpětné vyhledávání.* [2]

MG4J umožňuje indexaci velkých kolekcí dat. Kolekce se typicky skládá ze dvou a více souborů MG4J, jejichž formát je popsán v podkapitole 2.3. Invertovaný index je základní

[11] https://www.gnu.org/licenses/lgpl-3.0.en.html

myšlenkou celé indexace. Invertovaný index lze definovat, jako *seznam výskytů každého výrazu v textu* [5].

Prvním krokem indexace je skenování souborů celé kolekce, jehož výsledkem jsou dávkové soubory. Dávkové soubory lze považovat za dílčí indexy, které jsou podmnožinou dokumentů. Dávkové soubory jsou vytvářeny vždy, když počet zpracovaných dokumentů dosáhne uživatelem stanovenou hranici anebo pokud dojde k nedostatku paměti. Zjednodušeně je proces skenování popsán v tabulkách 3.1, 3.2 a 3.3.

Nejdříve jsou přiřazeny ukazatele k jednotlivým dokumentům.

Ukazatel	**Dokument**
0	I love you
1	God is love
2	Love is blind
3	Blind justice

Tabulka 3.1: Ukazatele dokumentů

Následně je vytvořen slovník, který obsahuje indexy všech výrazů v dokumentech.

Index výrazu	**Výraz**
0	blind
1	god
2	i
3	is
4	justice
5	love
6	you

Tabulka 3.2: Slovník indexů všech výrazů

V posledním kroku jsou vytvořeny trojice (v,d,p), které jsou definovány jako:

- **v – index výrazu**
- **d – index dokumentu**
- **p – pozice výrazu v dokumentu**

Výskyt výrazů ve stejném pořadí v jakém byly nalezeny
(2,0,0) (5,0,1) (6,0,2) (1,1,0) (3,1,1) (5,1,2) (5,2,0) (3,2,1) (0,2,2) (0,3,0) (4,3,1)

Tabulka 3.3: Výskyt výrazů

Názorným příkladem je první trojice**(2,0,0)**, kde můžeme vyčíst, že výraz s indexem 2 („*i*“) je v dokumentu s indexem 0 na pozici 0.

Posledním krokem je vytvoření invertovaného seznamu, který se vytvoří seřazením výskytů s rostoucím pořadím výrazů, pro příklad je znázorněna tabulka 3.4.

Výraz	Výskyt
0 (blind)	(0,2,2) (0,3,0)
1 (god)	(1,1,0)
2 (i)	(2,0,0)
3 (is)	(3,1,1) (3,2,1)
4 (justice)	(4,3,1)
5 (love)	(5,0,1) (5,1,2) (5,2,0)
6 (you)	(6,0,2)

Tabulka 3.4: Invertovaný seznam

Výsledkem procesu jsou dávkové soubory (subindexy), které se sloučí do jednoho indexu pomocí jedné ze tří metod:

- **Konkatenace** – Jde o vložení subindexů za sebe do jednoho indexu se změnou číslování dokumentů (u prvního dokumentu druhého subindexu se změní číslování indexu na o jedno větší než je index posledního dokumentu v prvním subindexu atd.).
- **Sloučení** – Tato metoda předpokládá, že subindexy byly již přečíslovány, tudíž je sloučí bez změny jejich indexů. Pokud nalezne dva dokumenty se stejným indexem, dojde k zastavení operace.
- **Vložení** – Jedná se o druh kombinace subindexů, který se používá pro virtuální dokumenty. Zjednodušeně řečeno, lze tuto metodu přirovnat k UNIXovému příkazu `paste`[12].

3.5 Dotazování nad MG4J

Dotazování nad zaindexovanými daty MG4J je důležité z hlediska vyhledání a zobrazení dat v uživatelsky příjemné formě. Tato část není využita v řešení této bakalářské práce, ale je využita ve vyhledávacím (a anotačním) nástroji K. Grešové, jehož backendem je dále popisovaný systém pro aktualizaci anotovaných dat.

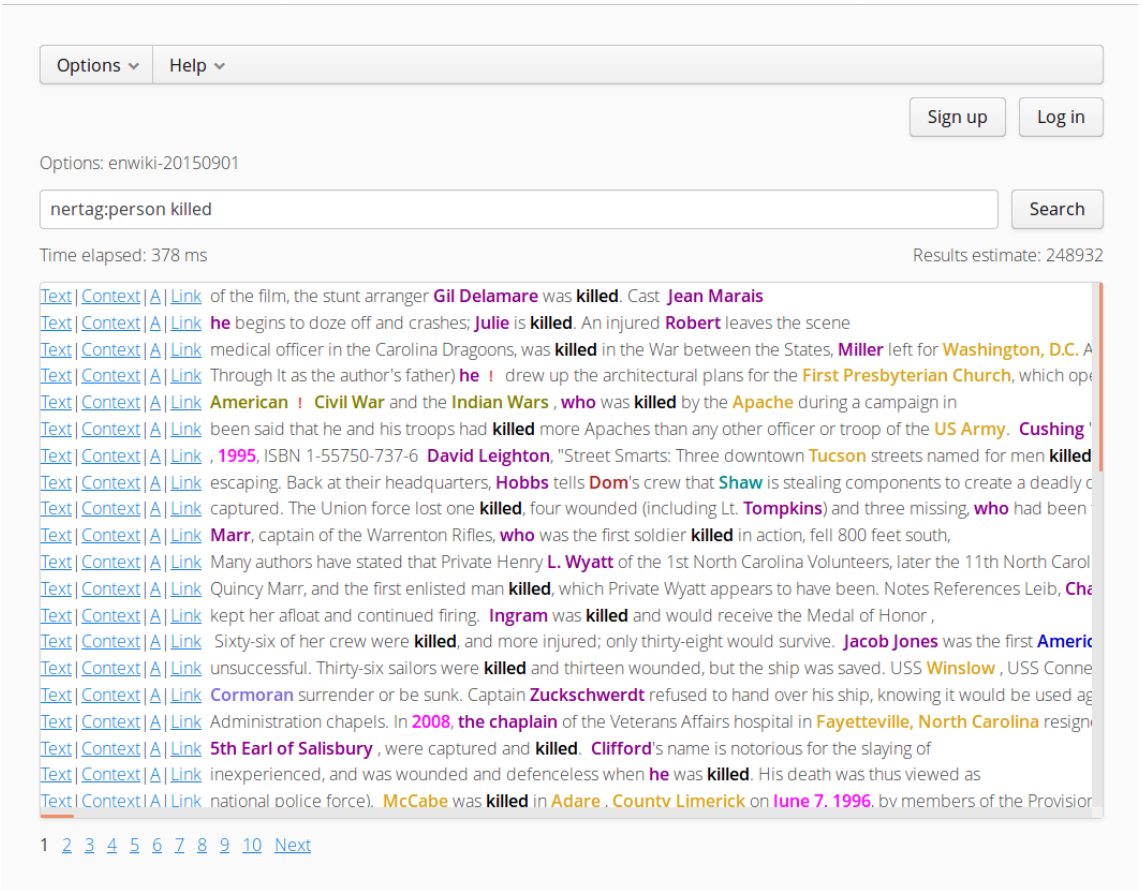
Na tato data se lze dotazovat pomocí uvozovacího výrazu (např. „*Computer*“). Výsledkem dotazu budou všechny dokumenty, které obsahují výraz „*Computer*“. Pro složitější dotazování lze využít operátory, přičemž lze využít i více operátorů najednou. Zde je uvedeno několik příkladů operátorů:

- **AND** – Konjunktivní vyhledání výrazu (např. „*Dog AND Cat AND Snake*“), výsledkem budou všechny dokumenty, které obsahují všechna tři zadaná slova.
- **OR** – Disjunktivní vyhledání výrazu (např. „*Dog OR Cat OR Snake*“), výsledkem budou všechny dokumenty, které obsahují alespoň jedno ze zadaných slov.
- **NOT** – Negace vyhledávání výrazu (např. „*NOT Dog*“), výsledkem dotazu budou všechny dokumenty, které neobsahují slovo „*Dog*“.
- **phrase** – Vyhledání části textu (např. „*"Dog and Cat"*“), výsledkem budou všechny dokumenty, které obsahují tuto část textu.

[12] https://linux.die.net/man/1/paste

3.6 Dotazovací nástroj

Pro vyhledání a zobrazení korpusových dat se v případe výzkumné skupiny KNOT využívá dotazovací nástroj MG4J GUI (graphical user interface). Autory nástroje jsou Jan Kouřil a Katarína Grešová. Dotazovací nástroj je dostupný na serveru athena1[13].



Obrázek 3.1: Dotazování v dotazovacím nástroji

Vyhledávání dokumentů je obdobné, jako u dotazování nad MG4J (popsáno v podkapitole 3.5). V obrázku 3.1 jsou v levé části vidět 4 tlačítka, která mají následující význam:

- **Text** – Po kliknutí se zobrazí okno, ve kterém se nachází celý text dokumentu. Při najetí na slovo zvýrazněné barvou se zobrazí zaindexované informace.
- **Context** – Po kliknutí se zobrazí nalezený úsek textu v širším kontextu.
- **A** – Po kliknutí se otevře anotační okno, kde se bude nacházet text celého dokumentu bez anotací (obrázek 3.2).
- **Link** – Obsahuje odkaz na zdroj textu.

V následujících příkladech jsou zobrazeny ukázkové dotazy, který jsou dostupné v oficiální dokumentaci projektu Corpora Processing Software [12].

[13] http://athena1.fit.vutbr.cz:8088/

Dotazy jsou automaticky přemapovány sémantickým indexem. To znamená, že není třeba zapisovat dotazy podobně jako v MG4J:

```
"(nertag:person{{nertag-> token}}) killed"
```

Ale lze je zapisovat následovně:

```
"nertag:person killed"
```

Dalším rozdílem oproti dotazování přímo nad MG4J je rozšíření Global constraints, které umožňuje označit token a provést na něm post-filtraci:

```
1:nertag:person < 2:nertag:person
1.fof != 2.fof AND 1.nerid = 2.nerid
```

Výsledkem dotazu jsou dokumenty, které obsahují stejnou osobu, ale v různé formě. Dalším operátorem je difference, který se používá pro dotazování v rámci jedné věty. Ukázka je následující:

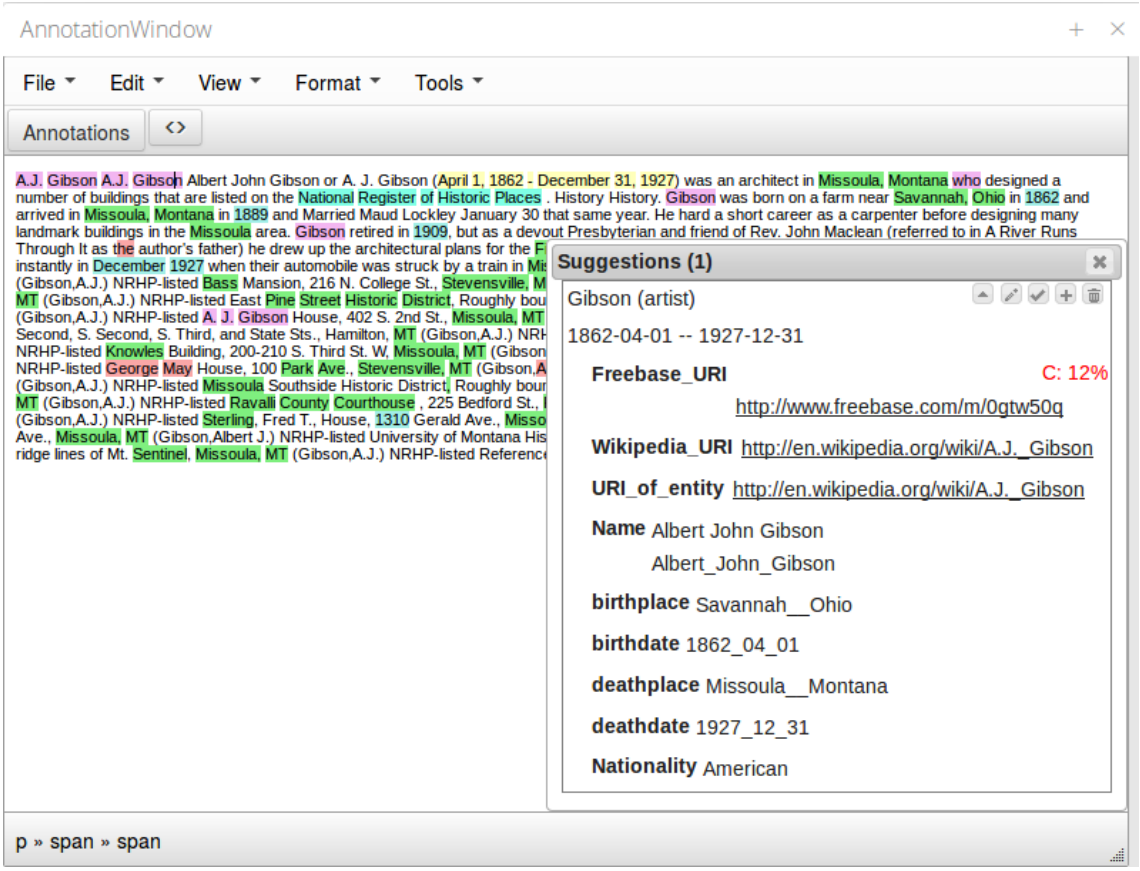
```
nertag:person < nertag:person - _SENT_
nertag:person < nertag:person - _PAR_
```

Výsledkem jsou dokumenty, které obsahují dvě anotace typu person v jedné větě (resp. odstavci). Při použití diference se vezme část textu, ve kterém se daný token nevyskytuje.

Dotazování nad indexy úzce souvisí s formátem MG4J. Například se lze dotazovat na všechny zaindexované sloupce anotací, které jsou zobrazeny v příloze A.1. Celý seznam indexů, nad kterými se lze dotazovat, je dostupný v oficiální dokumentaci projektu Corpora Processing Software[14].

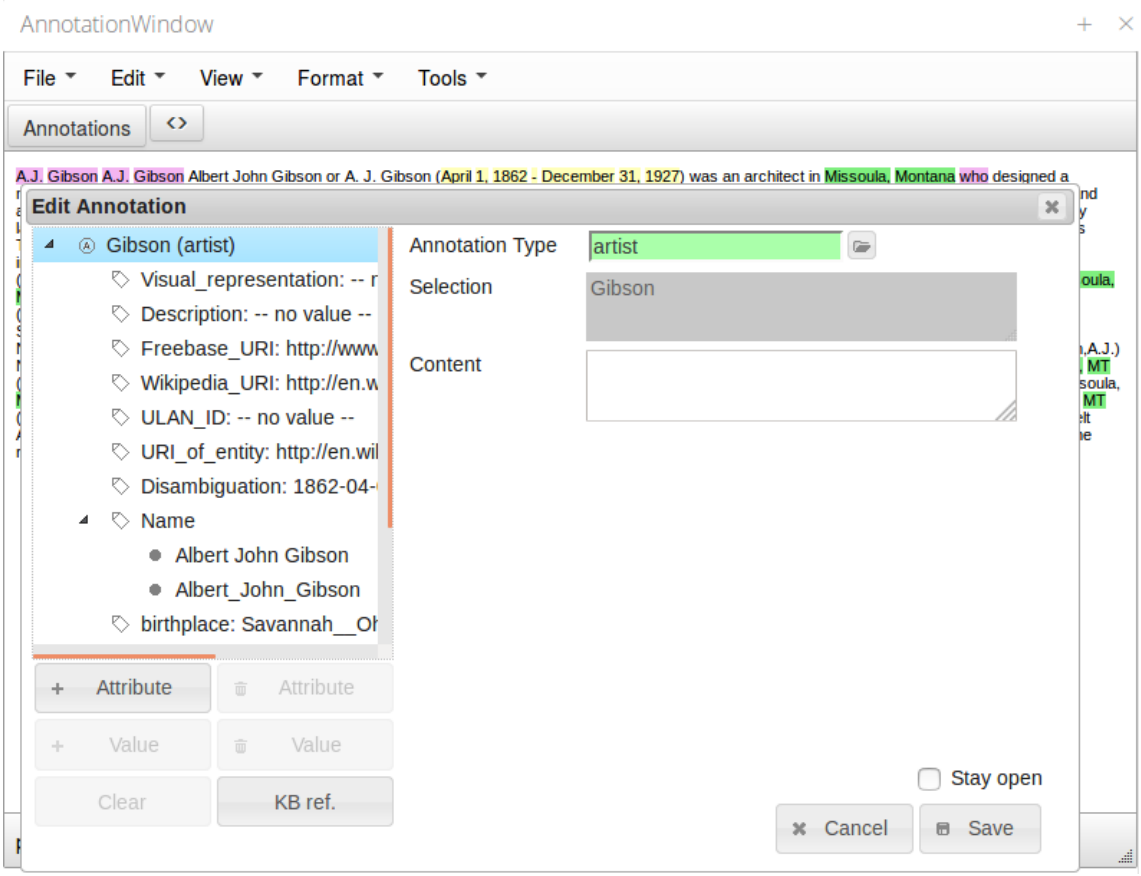
Na obrázku 3.2 se nachází názorná ukázka zobrazení anotací v textu, které se zobrazí při kliknutí na tlačítko **Annotations**. Typy anotací jsou odlišeny barvami zvýraznění. Také je zobrazeno okno anotace, které se zobrazí při najetí či kliknutí na anotované slovo (v příkladu slovo „*Gibson*“). Anotační okno zobrazuje všechny atributy anotace a umožňuje přejít na její editaci.

[14] http://knot.fit.vutbr.cz/corppproc/corppproc_cs.html#Indexy



Obrázek 3.2: Zobrazení anotací v anotačním nástroji

V následujícím obrázku 3.3 je zobrazena editace anotace slova „*Gibson*“. Uživatel zde může přidávat a měnit atributy.



Obrázek 3.3: Editace anotace v anotačním nástroji

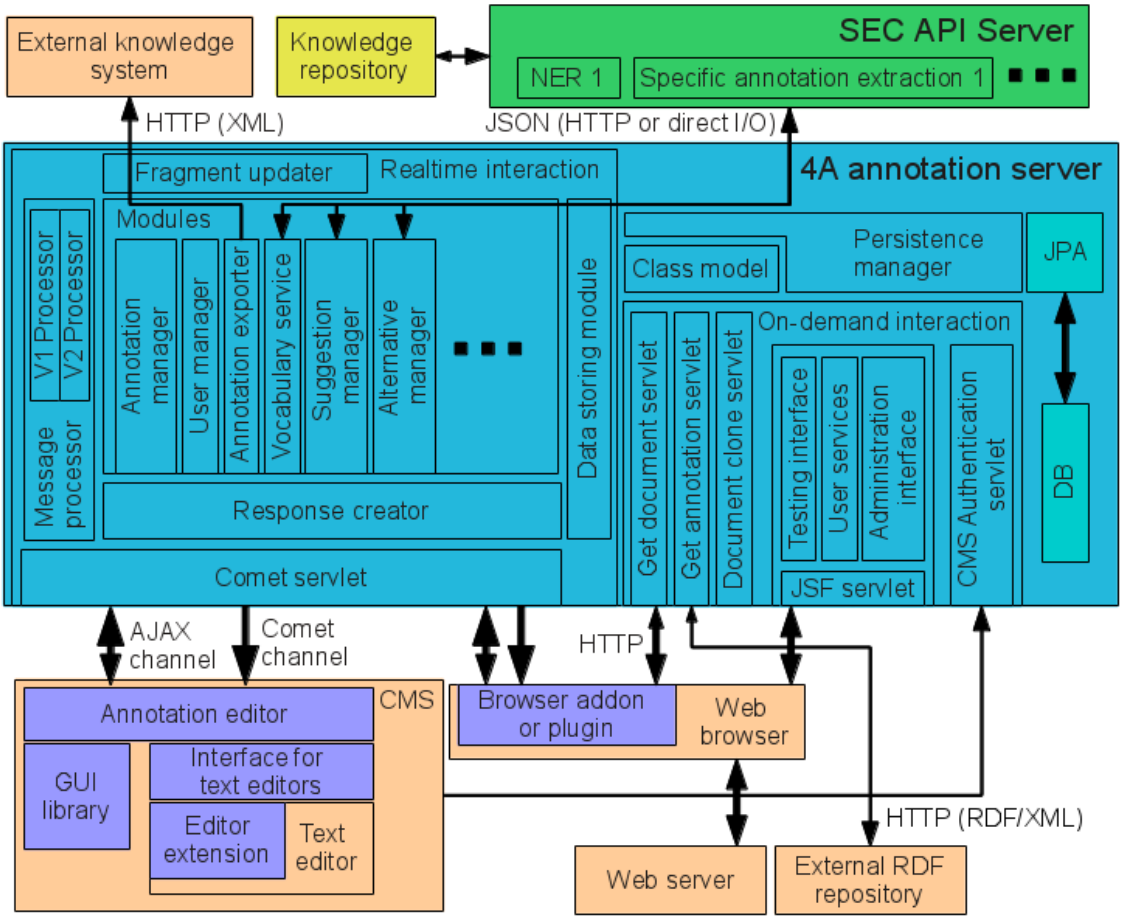
3.7 Anotační nástroj 4A

Annotations Anywhere, Annotations Anytime[15] (4A) je nástroj vytvářený výzkumnou skupinou KNOT a je vyvíjen od roku 2010. Nástroj 4A umožňuje anotovat texty, editovat anotace, ukládat anotace, sdílet anotace a další operace související s anotací textu. Nejdůležitější části systému jsou:

- **SEC API server** – generuje anotační návrhy,
- **Anotační server** – ukládá sémantické anotace a poskytuje služby klientům,
- **Klient** – umožňuje uživatelům vizuálně zobrazit anotace a editovat je.

Celé schéma systému je zobrazeno v obrázku 3.4. Dle schématu systém 4A využívá nástroj SEC, který sémanticky obohacuje text. Detailnější popis nástroje SEC je v podkapitole 3.2.

[15] http://knot.fit.vutbr.cz/annotations/



Obrázek 3.4: Schéma architektury systému 4A [11]

Systém podporuje dva rozdílné typy klientů. První z nich jsou ve formě rozšíření pro webové prohlížeče, které umožňují anotaci existujících webových stránek. V tuto chvíli je dostupné rozšíření pouze pro prohlížeč Mozilla Firefox. Další rozšíření pro ostatní prohlížeče jsou ve vývoji. Druhým typem klientů jsou editory, které umožňují vytvářet anotace a pozměňovat samotný text dokumentu. Tyto editory jsou označovány jako WYSIWYG (What you see is what you get). Zjednodušeně řečeno se jedná o editory, kde verze dokumentu zobrazená při editace je totožná s výslednou verzí dokumentu. V tuto chvíli existuje rozšíření verze 2 pro editor TinyMCE[16] [17], které je využito ve výše popsaném dotazovacím nástroji (uživatelské rozhraní klienta je na obrázcích 3.2 a 3.3).

Důležité je zmínit, že systém 4A ukládá anotace na server, kde jsou dostupné pro všechny uživatele systému. V tom je zásadní rozdíl od systému pro aktualizaci anotací v korpusech, kde se aktualizované anotace ukládají přímo do korpusu.

Velice obdobným konkurenčním řešením pro systém 4A je RDFaCE[17], který na rozdíl od systému 4A nepodporuje spolupráci více uživatelů v reálném čase. Další systémem je Pundit[18], který oproti 4A nepodporuje tvorbu nových atributů a proces anotace je výrazně pomalejší [17].

[16] https://www.tinymce.com/
[17] http://aksw.org/Projects/RDFaCE.html
[18] http://thepund.it/

Kapitola 4

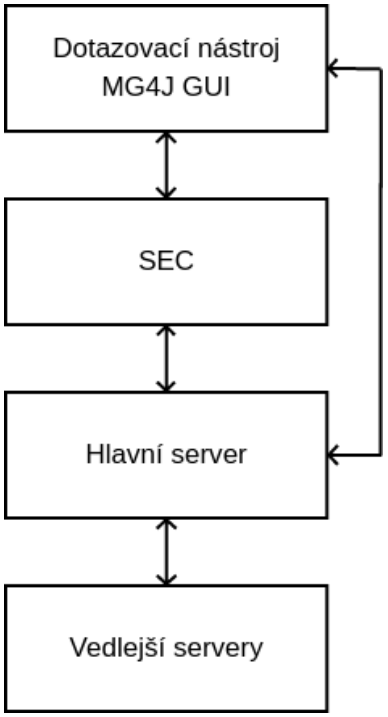
Návrh řešení aktualizace korpusových dat

Tato kapitola popisuje návrh systému pro aktualizaci anotací v korpusech. V návrhu se nejdříve zaměřím na obecný návrh celého systému. V dalších podkapitolách budou návrhy jednotlivých komponent systému.

Celý systém je navržen tak, aby podporoval dva rozdílné způsoby aktualizace. Pojmenoval jsem je jako normální a verzovací mód systému. Oba tyto módy systému nebudou spolu kompatibilní. Nejsou navrženy tak, aby šlo mezi nimi přepínat. Jejich hlavním rozdílem je způsob vzniku inkrementálního souboru ve formátu MG4J a následné zaindexování dat. Tyto rozdíly budou popsány v následujících podkapitolách.

V obrázku 4.1 jsou znázorněny 4 základní komponenty celého systému. Prvním z nich je dotazovací nástroj (popsaný v 3.6), který uživateli umožní vyhledat dokument a provést aktualizaci dokumentu. Druhá část systému je nástroj SEC, který bude sloužit k získání anotace vyhledaného dokumentu. V třetí a čtvrté části systému jsem se rozhodl využít serverové architektury. Řídícím prvkem bude hlavní server, který bude obsluhovat všechny vedlejší servery. Tento server bude přijímat požadavky od anotačního nástroje a přeposílat je na konkrétní vedlejší server. Mezi základní služby bude patřit získání textu dokumentu, získání anotací z MG4J pro SEC a aktualizace MG4J. Funkcí vedlejších serverů bude spouštění skriptů pro získání textu a anotací. Výstupy těchto skriptů se budou přeposílat na hlavní server, který je přepošle zpět anotačnímu nástroji nebo nástroji SEC. Poslední funkce vedlejších serverů bude vytváření inkrementálního souboru MG4J, který bude sloužit k aktualizaci a indexaci kolekcí MG4J.

V poslední podkapitole se budu zabývat stručným návrhem aktualizace KB. Tento skript nebude přímou součástí celého systému, spíše se jedná o rozšíření řešení celého problému.

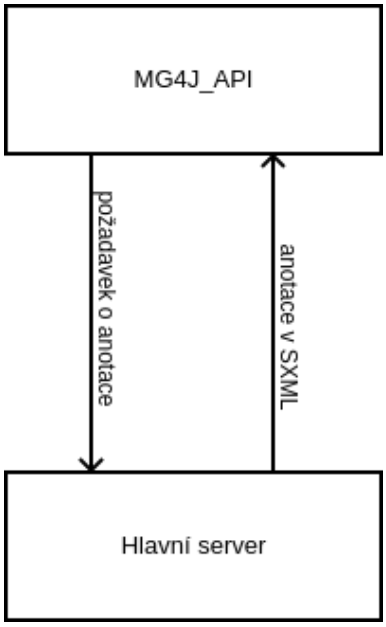


Obrázek 4.1: Obecný model systému

4.1 Návrh modulu pro SEC

Anotace v MG4J jsou omezené pouze na 8 sloupců, které mohou obsahovat jejich atributy. Proto je vhodné využít nástroj SEC, který umožňuje doplnit zbývající data ze znalostní báze a spojit anotace z MG4J s anotacemi získanými odlišným způsobem. Současně je nástroj SEC využíván anotačním nástrojem, takže nebude potřeba vytvářet nové rozhraní. Aby však mohl anotační nástroj využívat služby SEC k získání anotací z MG4J, je třeba vytvořit modul NER, který získá anotace z MG4J a spojí je s daty z KB.

Účelem vytvořeného modulu NER bude získat anotace zadaného dokumentu z MG4J, zpracovat je a doplnit o data z KB. Získání anotací se provede odesláním požadavku na hlavní server. Hlavní server zpracuje požadavek a odešle odpověď zpět do daného modulu, a to buď anotace ve formátu SXML, nebo příslušné chybové hlášení (např. nenalezení dokumentu apod.). Návrh komunikace mezi hlavním serverem a MG4J_API je zobrazen na obrázku 4.2.



Obrázek 4.2: Model komunikace mezi MG4J_API a hlavním serverem

Dalším krokem bude zpracování anotací ve formátu SXML a převod do formátu, který je kompatibilní s nástrojem SEC. Současně budou anotace doplněny o příslušná data z KB. Výsledkem procesu budou anotace ve formátu kompatibilním se vstupem SEC, který následně odešle odpověď (anotace) anotačnímu nástroji.

4.2 Hlavní server

Hlavní server bude plnit funkci prostředníka mezi vedlejšími servery a klientem a to z důvodu menšího zatížení sítě při vytváření dotazů anotačním serverem, resp. nástrojem NER (hlavní server se může dotázat pouze serveru, na kterém jsou požadovaná data). Současně poskytuje výhodu z hlediska zabezpečení, neboť vedlejší servery nemusí být přístupné z vnější sítě. Při návrhu serveru je třeba zohlednit všechny potřebné služby, které by tento server měl poskytovat. Pro potřeby anotačního nástroje a nástroje NER jsem navrhl následující služby:

- navrátit server, který vlastní zadaný URI,
- poskytnout celý text dokumentu zadaného pomocí URI, nebo jeho část,
- poskytnout všechny anotace k dokumentu zadanému pomocí URI, nebo jejich část,
- aktualizovat soubor MG4J danou anotací,
- aktualizovat soubor MG4J danou anotací ve verzovacím módu,
- provést dávkovou aktualizaci souborů MG4J,
- poskytnout seznam ID všech zastaralých dokumentů.

Server by měl podporovat zpracování více dotazů najednou. Proto byl zvolen konkurentní server. Pro komunikaci mezi servery bude využit protokol HTTP (Hypertext Transfer Protocol).

Dávková aktualizace bude služba, která bude sloužit k aktualizaci jedné anotace ve více dokumentech. Tato služba předpokládá, že v anotaci ve formátu SXML bude přidán seznam všech URI dokumentů, kterých se aktualizace týká. Hlavní server bude zpracovávat předaný seznam a přepošle jednotlivé požadavky dávkové aktualizace na konkrétní servery, které vlastní URI ze seznamu.

Poskytování verzovacích dokumentů je vysvětleno v podkapitole 4.3.

4.3 Vedlejší server

Vedlejší servery budou mít za úkol poskytovat a aktualizovat data z MG4J. Důvodem zavedení více vedlejších serverů je decentralizace dat a menší zatížení jednotlivých serverů. Služby, které jsem navrhl pro tento typ serverů, jsou obdobné jako u hlavního serveru:

- odpovědět na dotaz, zda vlastní zadané URI,
- poskytnout celý text dokumentu zadaného pomocí URI, nebo jeho část,
- poskytnout všechny anotace k dokumentu zadanému pomocí URI, nebo jejich část,
- aktualizovat soubor MG4J danou anotací ve formátu SXML,
- aktualizace MG4J – verzovací mód,
- provést dávkovou aktualizaci souborů MG4J,
- poskytnout verzovací soubor.

Na rozdíl od hlavního serveru, vedlejší server nebude přeposílat dotazy na jiný server, ale bude je zpracovávat. Pro zpracování MG4J bude mít každý server skript, který bude schopen nalézt dokument pomocí zadaného URI v MG4J a získat z tohoto dokumentu text či anotace. Dále bude mít dostupný skript, který bude vytvářet inkrementální soubor MG4J. Oba tyto skripty bude server spouštět samostatně. Poslední skript bude provádět indexaci celé kolekce nebo inkrementálního MG4J v závislosti na využívané verzi systému.

Inkrementální MG4J bude soubor, který bude obsahovat aktualizované dokumenty. Tento soubor bude ve formátu MG4J a bude se zvětšovat s počtem aktualizovaných dokumentů.

Verzovací soubor bude obsahovat stará ID aktualizovaných dokumentů. Server bude tento soubor poskytovat ve formátu JSON. Důvod zavedení verzovacího souboru je popsán v následujících dvou podkapitolách 4.6 a 4.7.

4.4 Ovládání vedlejších serverů

Pro jednodušší ovládání serverů jsem navrhl skript, který bude schopný provádět tři operace se servery. Tyto funkce jsou:

- spuštění všech serverů,

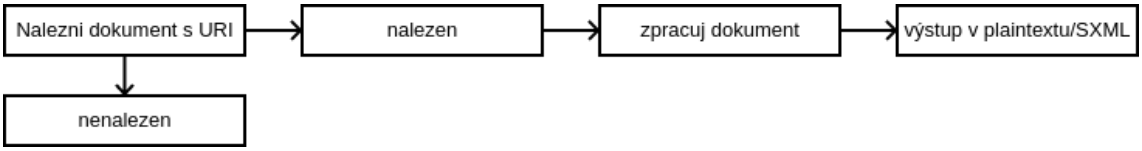
- restartování serverů,
- zastavení serverů.

Při spuštění serverů se budou vytvářet screeny[1] ve kterých budou spuštěny servery. Restartování vypne a zapne servery. Zastavení serverů ukončí činnost všech serverů a uzavře screeny.

4.5 Extrahování dat z MG4J

Základním stavebním prvkem celého systému je získání dat z MG4J. Uživatel nejdříve musí vidět, jaká data bude aktualizovat. Proto jsem navrhl skript, který v souboru MG4J vyhledá dokument pomocí URI tohoto dokumentu. Celý dokument by měl zpracovat a vytvořit jeden ze dvou výstupů a to text dokumentu, nebo anotace textu v daném dokumentu.

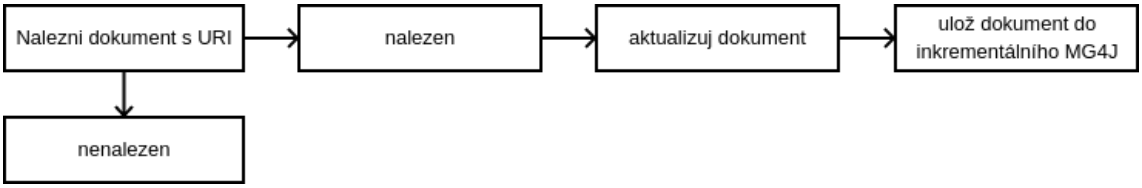
V návrhu skriptu je třeba zohlednit formát souborů MG4J (viz 2.3). Proto se v případě extrahování textu bude zpracovávat pouze sloupec č. 2 a při extrahování anotací se budou zpracovávat sloupce č. 14 až 25. Jednoduchý návrh modelu je na obrázku 4.3.



Obrázek 4.3: Jednoduchá logika extrahování dat z MG4J

4.6 Aktualizace anotačních dat ve formátu MG4J

Pro aktualizaci anotačních dat jsem navrhl skript, který bude aktualizovat dokumenty v souboru daty MG4J ze vstupního SXML. Aktualizovaný dokument se bude ukládat do inkrementálního souboru MG4J, který bude později sloužit k zaindexování aktualizovaných dokumentů. Model tohoto skriptu je na obrázku 4.4.



Obrázek 4.4: Jednoduchá logika aktualizace dat MG4J

Aktualizační skript bude mít podporu tří módů. Normální mód bude vytvářet pouze inkrementální soubor. Naopak verzovací mód bude přidávat verze dokumentu k ID dokumentů v inkrementálním MG4J. ID zastaralých dokumentů se budou ukládat do verzovacího souboru, který slouží jak k indexaci, tak i vedlejšímu serveru.

Poslední mód bude sloužit k dávkové aktualizaci a bude aktualizovat pouze jednu anotaci ve více dokumentech.

[1] Interaktivní shelly https://www.gnu.org/software/screen/

4.7 Zaindexování inkrementálního MG4J

Posledním krokem celé aktualizace bude zaindexování aktualizovaných dokumentů. Nejjednodušší by bylo provést indexaci pouze pro nově aktualizované dokumenty a zneplatnit staré dokumenty. Bohužel tato funkce není v momentální době podporována, což mi po e-mailové komunikaci potvrdil jeden z autorů systému MG4J, Sebastiano Vigna Ph.D. To znamená, že existují dva relevantní způsoby indexace. Prvním z nich je prosté přeindexování celé kolekce s aktualizovanými dokumenty. Druhým způsobem je zaindexování inkrementální souboru. V takovém případě je třeba udržovat seznam se zastaralými dokumenty, který se využije pro filtrování výsledků dotazů nad kolekcemi. To je hlavní důvod zavedení verzovacího módu systému.

Pro indexaci jsem se rozhodl navrhnout dva skripty. Jeden skript bude pro normální mód a druhý pro verzovací mód systému. Tyto skripty se budou spouštět v okamžiku, kdy se počet dokumentů v inkrementálním MG4J uzná jako dostatečný pro indexaci.

První skript bude pro normální mód aktualizace, kde bude docházet k vytváření inkrementálního souboru bez změny ID dokumentu. To znamená, že se bude zaindexovávat celá MG4J kolekce bez inkrementálního souboru. Před samotnou indexací se přepíší staré dokumenty aktualizovanými dokumenty z inkrementálního souboru MG4J. Poté se provede indexace, která zaindexuje celou kolekci. Tento způsob je jednoduchý a bezproblémový, ale jeho úskalí spočívá v časové náročnosti indexace celé kolekce, protože se budou zaindexovávat již zaindexované dokumenty.

Druhý skript pro indexaci jsem navrhl tak, že se bude zaindexovávat pouze inkrementální soubor MG4J, který se tak stane plnohodnotným souborem v nové kolekci v distribuovaném indexu. Tento způsob indexace je výrazně méně náročný oproti prvnímu, ale jeho problém spočívá v tom, že bude zaindexováno více verzí dokumentů v celém distribuovaném indexu. V takovém případě dotazovací nástroj vyhledá všechny verze dokumentů. To je důvod, proč bude při aktualizaci vznikat i verzovací soubor. Tento soubor bude dostupný dotazovacím serverům, které budou moci odstranit staré verze dokumentů z výsledků hledání.

4.8 Aktualizace znalostní databáze

Znalostní databáze se načítá do paměti při každém spuštění nástroje SEC. To znamená, že je vhodné aktualizovat KB jednorázově a v největší možné míře. V tomto směru mi velice pomohla data od Davida Prexty, který pracuje na identifikaci chyb v korpusech. Jeho data jsou uložena ve formátu TSV. Každý řádek obsahuje informaci o jedné entitě. Jednotlivé sloupce obsahují informace o názvu entity, odkazu na Wikipedii apod. Pro návrh skriptu jsou důležité sloupce 9 a 10, které definují typ chyby. Aktualizační skript by měl zpracovat data obsahující chyby a řešení těchto chyb a provést aktualizaci pro celou KB.

Kapitola 5

Použité technologie

Tato kapitola obsahuje popis všech důležitých technologií, které budou využity v implementaci jednotlivých částí systému.

5.1 Python

Python je vysokoúrovňový skriptovací programovací jazyk, který v roce 1991 navrhl Guido van Rossum. Je volně dostupný a přenositelný. Výkonost Pythonu spočívá v dynamickém typování a automatické správě paměti. Mezi jeho hlavní přednosti patří obrovská komunita vývojářů, kteří sdílí a tvoří volně dostupné knihovny [1].

Python jsem zvolil jako programovací jazyk, ve kterém bude vytvořen celý systém, a to z důvodu velké dostupnosti knihoven vhodných pro tuto práci a z důvodu zkušenosti s tímto jazykem.

V době psaní této práce byla nejnovější a stabilní verze Pythonu 3.6.1, ale pro řešení práce jsem se rozhodl využít verzi 2.7., která je nastavena jako výchozí na serverech výzkumné skupiny KNOT a bude výchozí i po nejbližším přechodu na novější verzi využívané distribuce operačního systému.

5.2 HTTP

Protokol HTTP (HyperText Transfer Protocol) je protokolem, kterým mezi sebou komunikují klienti a servery systému WWW. Protokol HTTP je v současné době implementován pouze nad protokolem TCP/IP, ale obecně vystačí s libovolným protokolem, který poskytuje spojované služby. První verze protokolu vznikla při vývoji systému WWW ve středisku CERN. [14]

Důležitá je znalost metod dotazu. V řešení práce bude využita metoda GET, která obecně slouží k vyzvednutí objektu (text, obrázek apod.). Při dotazu metodou GET se obvykle nevyužívá tělo dotazu. Druhou použitou metodou je POST, která se používá k odesílání dat v těle dotazu na server. Poslední verzí je HTTP 2.0, která byla standardizována RFC7540 v roce 2015 [3]. Protokol HTTP bude využit při komunikaci s hlavním serverem a při komunikace mezi hlavním serverem a vedlejšími servery. Oba tyto servery budou schopné zpracovávat pouze dotazy protokolem HTTP.

5.3 XML

XML (Extensible Markup Language) je značkovací jazyk, který je definován standardem REC-xml[1]. Tvůrcem tohoto jazyka je konsorcium W3C[2]. XML je užitečný především pro výměnu dat, která nepožadují definici vzhledu. V době psaní této práce existovaly dvě verze a to 1.0 a 1.1. Jejich hlavní rozdíl spočívá v používání znaků v názvech elementů, atributů atd. [7]. V řešení je využita verze 1.1.

5.4 SXML

SXML (Suggestions XML) je jazykem využívajícím jazyk XML pro popis anotací a nabídek anotací. SXML je definován pomocí gramatiky. Gramatika SXML, která se využívá ve výzkumné skupině KNOT, je popsána ve veřejné dokumentaci nástroje SEC[3].

Vzhledem k tomu, že je SXML založený na XML, jej lze jednoduše analyzovat pomocí dostupných parserů. Typickým příkladem je knihovna lxml, která je popsána v další podkapitole. Další výhodou je zabezpečení přenosu dat a podpora tvorby vlastních datových typů.

Typickým rozdílem mezi XML a SXML v nástroji SEC je rozdílný obsah výstupu. Výstup v XML obsahuje všechen text dokumentu, jak anotovaný, tak i neanotovaný. Naopak výstup v SXML obsahuje pouze anotovaný text.

SXML je využit v komunikaci mezi nástrojem SEC a anotačním nástrojem. SXML se bude využívat i v MG4J_API.

5.5 lxml

Knihovna lxml je využívána pro zpracování, editování nebo vytváření souborů ve formátu XML. Její hlavní předností je kombinace rychlosti s jednoduchostí.

V implementaci bude knihovna lxml využita při převodu anotací ze souborů MG4J do formátu SXML a naopak při opačném procesu aktualizace souborů MG4J z anotací ve formátu SXML. Konkrétně bude využit modul `etree`, který je nadstavbou ElementTree API[4].

5.6 JSON

JSON (JavaScript Object Notation) je datový formát (resp. způsob zápisu dat), který je nezávislý na počítačové platformě. Je určen především pro přenos dat, která nevyžadují speciální formát. JSON je čitelný a jednoduše zapisovatelný člověkem. To znamená, že jej lze jednoduše analyzovat nebo strojově generovat [8].

Tvůrcem formátu JSON je Douglas Crockford, který jej představil kolem roku 2000. V roce 2014 byly sjednoceny zastaralé standardy do jednoho a to do RFC7159, který přesně popisuje formát JSON [6].

V implementaci je JSON využit například při přenosu verzovacích souborů.

[1] https://www.w3.org/TR/REC-xml/
[2] https://www.w3.org/
[3] http://sec.fit.vutbr.cz/sec_documentation_cs.html#SXML
[4] http://effbot.org/zone/element-index.htm

Kapitola 6

Implementace

Tato kapitola popisuje implementaci důležitých a řídících částí celého systému. Nejdříve je popsán modul NER pro SEC, který extrahuje anotace z MG4J. Další podkapitola se zabývá stavbou serverů a popisuje jejich podporované služby, které jsou základním stavebním kamenem systému. Dále jsou popsány skripty, které extrahují a aktualizují soubory MG4J. V další podkapitole se zabývám popisem implementace indexace aktualizovaných dat. Zde je popsán rozdíl mezi verzemi systému. Předposlední podkapitola obsahuje popis implementace ovládání celého systému. Poslední kapitola obsahuje popis implementace skriptu aktualizující znalostní databázi.

6.1 NER modul MG4J_API

MG4J_API jsem implementoval jako modul v jazyce Python. Prvním krokem implementace bylo redefinovat třídu `NERTemplate`. Tato třída má tři hlavní metody: `_process()`, `_start()` a `_end()`.

Pro řešení je postačující redefinice metody `_process()`, která provádí celý úkon získání anotace. Metody `_start()` a `_end()` slouží k alokaci (resp. uvolnění) zdrojů při spuštění (resp. zastavení) nástroje NER využitého v daném modulu.

Prvním krokem činnosti MG4J_API je získání dat z MG4J. K tomuto slouží služba `mg4j_get_sxml_with_offset`, která je popsána v podkapitole 6.2. Nejdříve se pošle požadavek na hlavní server, který odpovídá anotacemi ve formátu SXML. Tyto anotace se zpracují a uloží do proměnných, které jsou kompatibilní s vnitřní reprezentací nástroje SEC. Speciálně jsou upravovány anotace typu `date` a `interval`, které musí být ve formátu ISO 8601[1]. U ostatních typů anotací dochází k uložení atributů do speciální proměnné datového typu seznam `ent_direct_attributes`. Posledním krokem je upravení výstupního formátu, který potřebuje SEC. Spojení anotací z MG4J_API s anotacemi z KB se provádí automaticky v nástroji SEC. To znamená, že nebylo třeba využít nástroj FIGA, který vrací číslo řádku entity v KB.

MG4J_API je v současné době nainstalován v nástroji SEC, který je spuštěn na serveru knot09[2]. SEC využije tento modul NER, pokud bude v požadavku nastavena hodnota `enrichment_engine` na `mg4j`.

[1] https://www.w3.org/TR/NOTE-datetime

[2] http://knot09.fit.vutbr.cz:8082/

6.2 Serverová architektura a její služby

Servery mají za úkol zprostředkovávat komunikaci mezi anotačním nástrojem a korpusem. Při implementaci serverů jsem se pokusil využít dostupné knihovny, které umožňují jejich jednoduchou implementaci (např. flask, BaseHTTPServer apod). Bohužel, zde jsem narazil na problém při zpracovávání požadavku typu GET. Problém spočíval v tom, že všechny vyzkoušené knihovny escapovaly URI, který měl sloužit k vyhledání dokumentu v MG4J. To vedlo k nenalezení dokumentu, protože některé znaky se nepřevedly zpět do původní podoby při použití metody `unquote()` dostupné v knihovně urllib. Proto jsem se rozhodl implementovat servery od základu za použití knihovny socket.

Nejdříve se vytvoří TCP/IP socket, ke kterému se přidruží host a port serveru. Poté byla implementována nekonečná smyčka, ve které socket čeká na příchozí spojení. Dalším krokem bylo zařadit do serverů konkurentnost. Té jsem docílil vytvořením nového procesu při každém spojení.

Posledním krokem bylo zpracování dotazu. Proto jsem vytvořil funkci `handle()`, která se volá ihned po vytvoření nového procesu. V této funkci se nejdříve přijmou veškerá data, poté se provede kontrola dotazu (zda se jedná o GET nebo POST) a oddělení hlavičky od těla zprávy. Následně dojde k určení typu služby a zavolá se příslušná funkce, které tuto službu definuje. Jakmile se funkce provede, dojde k vytvoření odpovědi a ukončení spojení.

Důležitou součástí jsou podpůrné soubory, které slouží ke konfiguraci a k optimalizaci některých úkonů serverů. Hlavní server například musí mít dostupný soubor, který obsahuje seznam všech serverů, které má obsluhovat. Vedlejší servery mají soubor, který obsahuje seznam všech URI dokumentů, které vlastní. Toto vede k optimalizaci a urychlení při využití některých služeb.

Nutno dodat, že jak hlavní, tak vedlejší server jsou implementačně velice podobné. Jejich hlavní rozdíl spočívá ve službách, které podporují. Před samotném popsáním služeb je třeba popsat parametry dotazů:

- **<uri>** – URI hledaného dokumentu,
- **<start_offset>** – počáteční offset části dokumentu,
- **<end_offset>** – koncový offset části dokumentu.

Hlavní server poskytuje následující služby:

- **/who_has_uri/<uri>** – Služba má za úkol najít vedlejší server, který vlastní URI. To znamená, že služba přeposílá dotaz `is_uri` na všechny vedlejší servery, které obsluhuje. Tento proces musí být rychlý a efektivní. Proto jsem využil vícevláknové implementace. Jedno vlákno odpovídá jednomu dotazu na vedlejší server. Po odpovědi od všech vedlejších serverů dochází ke zpracování odpovědí. Služba vrací adresu serveru, který odpověděl, že je vlastníkem zadaného URI.
- **/get_plain_text_with_offset/<start_offset>/<end_offset>/<uri>** – Služba, která vrací plaintext dokumentu z MG4J. Prvním krokem této služby je nalezení vlastníka URI. Proto využívá funkce `who_has_uri()`, která obsluhuje předchozí službu. Po nalezení se vytváří dotaz na konkrétní vedlejší server. Pokud nejsou zadané offsety (jejich hodnoty jsou nastaveny na 0), posílá se dotaz na službu vedlejšího serveru `get_plain_text`. Naopak, je-li nastaven alespoň jeden offset na nenulovou hodnotu, dojde k využití služby `get_plain_text_with_offset`. Službu využívá anotační nástroj.

- **/mg4j_get_sxml_with_offset/<start_offset>/<end_offset>/<uri>** – Služba, která vrací anotace dokumentu ve formátu SXML. Proces této služby je obdobný jako u předchozí služby. Nejdříve se zjistí vlastník dokumentu, poté se provede kontrola offsetů. Posledním krokem je vytvoření dotazu na konkrétní vedlejší server. Pro nulové hodnoty offsetů je dotaz `get_sxml`, pro nenulové dotaz `get_sxml_with_offset`. Tato služba je využita v MG4J_API.

- **/update_sxml/<uri>** – Služba, která přeposílá anotaci ve formátu SXML na vedlejší server, který je vlastníkem URI. Proces je opět podobný předchozím službám. Nejdříve zjistí, který server vlastní dokument, a poté přepošle dotaz na službu vedlejšího serveru `update_sxml`.

- **/update_sxml_version/<uri>** – Tato služba je stejná jako předchozí. Liší se pouze v tom, že nepřeposílá dotaz pomocí služby `update_sxml`, ale využívá službu `update_sxml_version`. Jejich rozdíl bude vysvětlen v popisu služeb vedlejších serverů.

- **/batch_update_sxml** – Tato služba slouží k dávkové aktualizaci. To znamená, že je využita pro aktualizaci jedné anotace, která se nachází ve více dokumentech. V momentální fázi je služba implementována tak, že předpokládá existenci seznamu URI, kterých se týká aktualizace v SXML. Proto se zpracuje SXML a vyhledá se element, který se jmenuje `uri_list`. Každý URI v tomto elementu je uložen a element je odstraněn z SXML. Poté se pro každý URI zjistí vlastník dokumentu a vytvoří se dotaz `update_sxml_batch` na příslušný server. V aktuální verzi dotazovacího a anotačního nástroje tato služba není podporována – její využití je naplánováno v blízké budoucnosti.

- **/get_version_file** – Služba, která vytváří dotaz `get_version_file` na všechny servery, zpracuje všechny odpovědi a sjednotí je do jednoho dokumentu ve formátu JSON, který následně vrací.

Hlavní rozdíl mezi službami vedlejšího a hlavního je ten, že služby u vedlejších serverů zpracují dotaz a spouští konkrétní skript vytvořený pro tuto službu. Popis služeb vedlejších serverů je následující:

- **/is_uri/<uri>** – Cílem služby je odpovědět hlavnímu serveru, zda je tento vedlejší server vlastníkem URI nebo ne. Proces u této služby je jednoduchý, pouze se otevře soubor, kde se nachází všechny URI, které server poskytuje, a porovnávají se s URI předaným dotazem. Služba na dotaz odpovídá řetězcem „*Found*“ (resp. „*Not found*“).

- **/get_plain_text/<uri>** – Služba, která slouží k získání textu dokumentu bez omezení offsety. Nejdříve sever zjistí ze souboru s URI, v kterém MG4J souboru se URI nachází. Poté se spouští skript `mg4j_convert.py`, který je popsán v podkapitole 6.3. Výstup skriptu je odeslán v těle odpovědi hlavnímu serveru.

- **/get_plain_text_with_offset/<start_offset>/<end_offset>/<uri>** – Tato služba je obdobná jako předchozí. Odlišuje se pouze omezením textu offsety. To znamená, že skript `mg4j_convert.py` je spuštěn s parametry `--start_offset` a `--end_offset`, které vymezují rozsah výstupu skriptu.

- **/get_sxml/<uri>** – Služba, která získává anotace zadaného dokumentu ve formátu SXML. V případě volání této služby dochází opět k zjištění souboru MG4J, který obsahuje URI. Poté se spouští skript `mg4j_convert.py` v módu, jehož výstupem jsou anotace dokumentu. Následně je tento výstup přeposlán na hlavní server v těle zprávy.
- **/get_sxml_with_offset/<start_offset>/<end_offset>/<uri>** - Služba je opět obdobná jako předchozí. Výstup skriptu `mg4j_convert.py` je omezen parametry `--start_offset` a `--end_offset`.
- **/update_sxml/<uri>** – Služba, která slouží k aktualizaci dokumentů v souborech MG4J. Prvním krokem v procesu této služby je vyhledaní URI. Poté se ukládá SXML do inkrementálního SXML souboru a do dočasného souboru. Posledním krokem je spuštění aktualizačního skriptu `mg4j_update.py`, který je popsán v podkapitole 6.3. Výsledkem procesu je odpověď hlavnímu serveru o úspěchu či neúspěchu aktualizace.
- **/update_sxml_version/<uri>** – Služba, která slouží k aktualizaci dokumentů v souborech MG4J ve verzovacím módu systému. Rozdíl oproti předchozí službě je v přidání přepínače `-v` při spuštění aktualizačního skriptu.
- **/update_sxml_batch/<uri>** – Služba, která provádí dávkovou aktualizaci dokumentů v souborech MG4J. Rozdíl oproti předchozím službám je opět pouze v použití přepínače `-b`.
- **/get_version_file** – Služba, která poskytuje soubor, který obsahuje stará ID dokumentů, které byly aktualizovány. Obsah souboru je převeden do formátu JSON a odeslán na hlavní server. Tuto službu lze využít pouze pokud je systém ve verzovacím módu.

V předchozím přehledu služeb nejsou uvedeny všechny dostupné služby, které servery poskytují, a to z důvodu, že nejsou využity pro fungování celého systému. Tyto služby lze využít k testování a ladění.

6.3 Práce se soubory MG4J

Jak bylo uvedeno v kapitole 4, extrakce dat ze souborů ve formátu MG4J a jejich následná aktualizace je důležitá pro funkčnost celého systému. Vytvořil jsem dva skripty v jazyce Python. Prvním z nich je `mg4j_convert.py` zmíněný v předchozí podkapitole. Tento skript umožňuje získat data z MG4J a to buď text dokumentu, nebo anotace dokumentu.

Skript jsem implementoval tak, že nejdříve dochází ke zpracování argumentů skriptu. Tyto argumenty jsem popsal v tabulce 6.1 popisující všechny funkce skriptu.

Argument	Funkce
`--file`	Udává cestu k souboru MG4J, který bude zpracováván. Parametr je povinný.
`--uri`	Značí URI dokumentu, který bude zpracován. Parametr je povinný.
`--start_offset`	Argument umožňuje vymezit rozsah výstupu. Značí počáteční offset rozsahu vymezení. Parametr je volitelný.
`--end_offset`	Argument umožňuje vymezit rozsah výstupu. Značí ukončovací offset rozsahu vymezení. Parametr je volitelný.
`-s`	Přepínač nastavuje výstup na standardní výstup (`stdout`) a to pouze anotací v SXML. Přepínač je volitelný.
`-t`	Přepínač nastavuje výstup na standardní výstup (`stdout`) a to pouze text dokumentu v plaintextu. Přepínač je volitelný.

Tabulka 6.1: Popis argumentů skriptu `mg4j_convert.py`

Poté se otevře soubor MG4J a skript iteruje nad každým řádkem souboru. Zde je třeba si uvědomit užitečnost znalosti formátu souboru MG4J. Jak bylo zmíněno v podkapitole 2.3, každý dokument je uvozen speciálními řádky.

Pro implementaci je důležitý řádek obsahující uvozovací řetězec „`%%#PAGE`“, který obsahuje URI dokumentu. Proto v hlavní smyčce skriptu dochází nejdříve ke kontrole, zda se shoduje URI v aktuálním řádku MG4J s URI předaným pomocí argumentu skriptu. Pokud se nerovnají, jsou nadcházející řádky ignorovány až do řádku značícího nový dokument. V případě shody se celý dokument zpracuje řádek po řádku. Zpracuje se jak text dokumentu, tak i anotace.

Při zpracování textu stačí pouze spojit všechny hodnoty ve sloupci č. 2 mezerou. Jediná operace, která se provádí s textem, je odstranění tzv. lepidel (značek `<g/>`), která určují, zda se má za daným slovem nacházet mezera či nikoliv.

Zpracování anotací je o něco složitější. Při zpracování každého řádku dochází ke kontrole sloupců č. 14 a 15. První ze zmíněných obsahuje ID anotace a druhý typ anotace. Pokud je sloupec s typem anotace nenulový, zavolá se se funkce `get_anot_type()`, která nejdříve určí typ anotace a poté volá příslušnou funkci k typu anotace. Tzn. pokud je typ anotace `person`, tak se zavolá funkce `person_xml()`, která má nadefinovaný význam sloupců č. 18 až 25. Výstupem této funkce je uzel SXML, který obsahuje všechny atributy, které byly v souboru MG4J nenulové.

V případě nulové hodnoty sloupce s typem, ale zároveň nenulové hodnoty sloupce s ID, se jedná o koreferenční anotaci. Proto je vytvořen uzel v SXML, který prozatím obsahuje pouze slovo, offsety a ID anotace.

Posledním krokem je doplnění všech koreferenčních uzlů o atributy anotací. Poté už zbývá připravit výstup skriptu. Dochází ke kontrole argumentů, které značí zda je výstup

skriptu text (resp. SXML anotace) a zda se výstup vypíše na standardní výstup nebo do souboru.

Aktualizační skript využívá stejné konstrukce jako skript pro extrakci dat. Nazval jsem ho `mg4j_update.py`. Proces celého skriptu je podobný, proto ho jen stručně zopakuji a uvedu rozdíly oproti `mg4j_convert.py`. Je třeba podotknout, že skript provádí aktualizaci ze souboru ve formátu SXML, který vytvořil vedlejší server. Pro lepší představu o funkcích skriptu jsem vytvořil tabulku 6.2, která obsahuje popis argumentů.

Argument	**Funkce**
`--file`	Udává cestu k souboru MG4J, který obsahuje dokument určený k aktualizaci. Parametr je povinný.
`--sxml_file`	Udává cestu k dočasnému souboru ve formátu SXML vytvořenému vedlejším serverem. Parametr je povinný.
`--uri`	Značí URI dokumentu, který bude zpracován. Parametr je povinný.
`--start_offset`	Argument umožňuje vymezit rozsah aktualizace. Značí počáteční offset rozsahu vymezení. Parametr je volitelný.
`--end_offset`	Argument umožňuje vymezit rozsah aktualizace. Značí ukončovací offset rozsahu vymezení. Parametr je volitelný.
`-b`	Zapíná dávkovou aktualizace. Přepínač je volitelný.
`-v`	Zapíná verzovací mód. Skript upraví verzi aktualizovaného dokumentu. Přepínač je volitelný.

Tabulka 6.2: Popis argumentů skriptu `mg4j_update.py`

Nejdříve se zpracují argumenty, poté SXML pomocí metody `parse()` z knihovny lxml. Následně se vyhledá dokument pomocí URI stejně jako u předchozího skriptu. Proces zpracování dokumentu je obdobný. Skript iteruje řádek po řádku.

Pro každé slovo se vypočítávají offsety a kontroluje se, zda se slovo s vypočítaným offsetem nachází v SXML. Pokud takový uzel v SXML neexistuje, řádek se zapíše do nově vzniklého inkrementálního MG4J. Pokud taková anotace existuje, zavolá se funkce `update_line()`. Tato funkce nejdříve určí typ anotace a volá funkci v závislosti na typu anotace. V takové funkci dojde ke zpracování uzlu a aktualizují se příslušné sloupce v řádku MG4J. Výstupem funkce je aktualizovaný řádek, který je zapsán do inkrementálního souboru MG4J. Výstupem celého skriptu je aktualizovaný dokument uložený v inkrementálním souboru.

Při dávkové aktualizaci se aktualizují všechny řádky, které obsahují anotaci v SXML. Předpoklad pro dávkovou aktualizaci je, že SXML bude obsahovat pouze jeden uzel. Pokud by obsahoval více uzlů (tzn. více rozdílných anotací), tak se aktualizace provede pouze pro první uzel.

Verzovací mód se ve skriptu promítne přidáním verze dokumentu k ID. Skript v takovém případě kontroluje verzovací soubor a dle poslední verze dokumentu upravuje ID a zapisuje staré ID do souboru.

6.4 Indexace

V kapitole 4 je popsán návrh dvou skriptů, které budou provádět poslední krok celé aktualizace. Oba skripty jsem implementoval přesně dle návrhu a podporují stejné argumenty uvedené v tabulce 6.3.

Argument	**Funkce**
`--list_of_uris`	Udává cestu k souboru, který obsahuje všechny URI vlastněné serverem. Parametr je povinný.
`--incremental_mg4j`	Udává cestu k inkrementálnímu souboru MG4J. Parametr je povinný.
`--indexing_path`	Udává cestu ke kolekci, která bude zaindexována. Parametr je povinný.
`--bad_files_path`	Udává cestu k adresáři, kde se budou ukládat špatně zaindexované soubory MG4J. Parametr je povinný.
`--final`	Udává cestu k adresáři, kde se uloží výsledek indexace. Parametr je povinný.

Tabulka 6.3: Popis argumentů indexovacích skriptů

Skripty jsou `inkremental_actualize.py` a `inkremental_actualize_version.py`. Jak názvy napovídají, první z nich slouží pro normální mód systému a druhý pro verzovací mód.

První ze skriptů nejdříve aktualizuje soubory MG4J z inkrementálího MG4J a to prostým přepisem starých dokumentů. Po aktualizaci všech dokumentů obsažených v inkrementálním MG4J dojde k přeindexování celé kolekce.

Druhý skript, určený pro verzovací mód, naopak nejdříve provádí změnu souboru předaného argumentem `--list_of_uris`. Upravuje v něm cestu k aktualizovaným dokumentům, které se nacházejí v inkrementálním MG4J. Po úpravě souboru dochází k zaindexování inkrementálního MG4J. Tím se inkrementální MG4J stane plnohodnotným souborem v nové kolekci v distribuovaném indexu.

6.5 Ovládání systému

Celý systém je postavený na komunikaci mezi různými servery. Proto jsem vytvořil skript `servers_handling.py`, který slouží k ovládání vedlejších serverů. Důležitým předpokladem pro správnou funkčnost skriptu je nastavení ssh klíčů pro každý server. Toho lze dosáhnout například pomocí příkazu `ssh-copy-id user@host`. Skript vyžaduje mimo jiné i argument `--servers_file`, který udává cestu k souboru, který obsahuje nastavení serverů. Tento soubor má 4 sloupce oddělené znakem tabulátoru.

První sloupec reprezentuje hostitelský server, druhý port serveru, třetí cestu k adresáři, ve kterém se nachází skript `slave_server.py`, a poslední sloupec reprezentuje cestu k souboru, který obsahuje všechny dostupné URI pro tento server. Seznam sloupců je v tabulce 6.4.

host	port	cesta k slave_server.py	cesta k uris_file

Tabulka 6.4: Popis formátu souboru obsahujícího nastavení serverů

Skript podporuje tři módy, které odpovídají přepínačům:

- `--start` – vytvoří na každém serveru screen, spustí server se zadaným nastavením a odpojí se od serveru.
- `--restart` – připojí se na každý server, navrátí se do screenu, zastaví server, znovu jej spustí a odpojí se.
- `--stop` – připojí se na servery, ukončí činnost serverů a uzavře screeny.

Poslední volitelný argument `--username` slouží k nastavení přihlašovacího jména. Například pokud uživatel spouští skript ze zařízení, na kterém má jiné přihlašovací jméno než na serverech, může být užitečný tento argument, který umožňuje specifikovat uživatelské jméno, které používá pro přihlášení na servery.

6.6 Skript aktualizující znalostní databázi

Skript aktualizující KB má velice jednoduchou posloupnost operací. V první fázi dochází ke zpracování dat od Dávida Prexty. U každého řádku dochází k určení typu chyby, a to z důvodu omezené možnosti aktualizací. Typy chyb, které lze opravit, jsou:

- REDIRECT MATCH – UPDATE WIKIPEDIA URI IN KB,
- MAP MATCH – EXPAND KB BY URI FROM DATASET.

První typ chyby označuje entity, u kterých je třeba aktualizovat URI, a to z důvodu zastaralého URI, který je přesměrován na nový. Druhý typ chyby označuje všechny entity, u kterých lze doplnit hodnotu do sloupce pro URI na Freebase, URI na Wikipedii, nebo URI na DBpedii.

Řádky s těmito chybami jsou ukládány do struktury, která obsahuje důležité informace o chybě (např. URI entity, řešení chyby apod.). Po zpracování je struktura vyfiltrována. Skript odstraní duplicitní záznamy chyb. Takové duplicitní záznamy mají identické sloupce kromě jednoho, a to sloupce obsahujícího informaci o kontextu. Taková informace je nepodstatná, tudíž skriptu stačí pouze jeden záznam.

Po odstranění duplicitních záznamů dochází k načtení celé KB do paměti a iterování nad každým řádkem KB. Pro každý řádek se kontroluje struktura, zda neobsahuje chybu o tomto řádku. Pokud ano, provede se aktualizace, nový řádek se zapíše do souboru a záznam o chybě se odstraní ze struktury. Pokud ne, nezměněný řádek se zapíše.

Data od Dávida Prexty obsahovala další typy chyb. Například záznamy o duplicitních entitách v KB. Bohužel tyto chyby skript neumí opravit a to z důvodu netriviálního rozhodování o entitě, která by měla v KB zůstat, a entitě, která by měla být smazána.

Kapitola 7

Testování a Experimenty

V této kapitole bude pomocí experimentů a testů prokázána funkčnost navrženého a implementovaného systému pro aktualizaci anotací v korpusech.

V první části se zabývám automatickým testováním služeb. Především jsem se zaměřil na měření rychlosti zpracování požadavků u jednotlivých služeb. Další podkapitola pojednává o ručním testování a kontrolování nově vzniklých inkrementálních souborů MG4J. V předposlední podkapitole se stručně zabývám popisem výsledku aktualizace znalostní databáze. V poslední podkapitole jsou popsány problémy, na které jsem narazil při testování a nebyly v době odevzdání technické zprávy vyřešeny.

Veškeré testování jsem provedl na menším z dostupných korpusů, a to na korpusu s názvem `enwiki-20150901`. Tento korpus jsem zvolil především z důvodu jeho dostupnosti ve webovém rozhraní MG4J GUI. Korpus `enwiki-20150901` obsahuje **4 929 887 dokumentů** a celková velikost dat je **231,9 GB**. Korpus je distribuován mezi **32 serverů**, kde **na každém** serveru je **v průměru** uloženo **7,2 GB** dat.

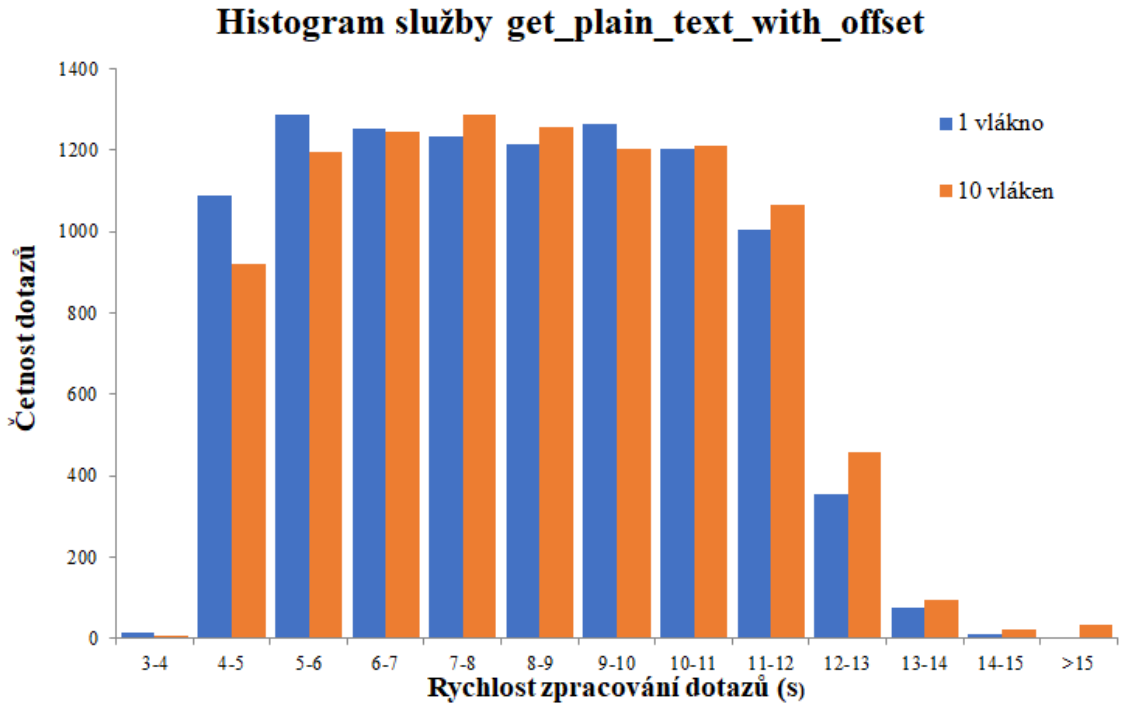
7.1 Rychlost služeb

Rychlost poskytnutí dat je důležitým aspektem pro uživatele systému. Proto jsem se rozhodl otestovat služby, které slouží k získání textu nebo anotací.

Otestoval jsem obě nejdůležitější služby hlavního i vedlejších serverů, kterými jsou `get_plain_text_with_offset` a `mg4j_get_sxml_with_offset`. Také jsem v testech zahrnul MG4J_API.

Postup u všech zmíněných testů je v této podkapitole stejný. Nejdříve jsem si vytvořil dokument, který obsahoval všechny URI korpusu `enwiki-20150901`. Poté jsem vytvořil pro každou službu skript, který pseudonáhodně vybral **10 000 URI** ze souboru a posílal požadavek na hlavní server o získání textu či SXML zadaného pomocí URI. Pro ověření konkurentní vlastnosti serveru jsem testy doplnil o vlákna. To znamená, že jsem na server odesílal **10 paralelních požadavků** pro různé URI. Následující histogramy zobrazují četnost dotazů v závislosti na rychlosti jejich zpracování.

První histogram, který je na obrázku 7.1, představuje data získaná z testů služby `get_plain_text_with_offset`. Z histogramu vyplývá, že při posílání **10 dotazů** na hlavní server naráz, došlo k menšímu nárůstu doby zpracování oproti době zpracování **1 dotazu**. Tato změna je nejvíce vidět při hodnotách četnosti pro **4–5 a 12–13 sekund**. Také lze z histogramu vyčíst zvýšenou četnost u doby zpracování **15 a více sekund**.



Obrázek 7.1: Histogram četnosti dotazů v závislosti na rychlosti zpracování služby `get_plain_text_with_offset`

Druhý histogram, který je umístěn na obrázku 7.2, zobrazuje výsledky testů pro službu `mg4j_get_sxml_with_offset`. Stejně jako v předchozím případě jsem do histogramu vložil výsledky posílání **1 dotazu** na hlavní server a výsledky posílání **10 dotazů** na hlavní server naráz. Také jsem přidal výsledky testování MG4J_API, které tuto službu využívá.

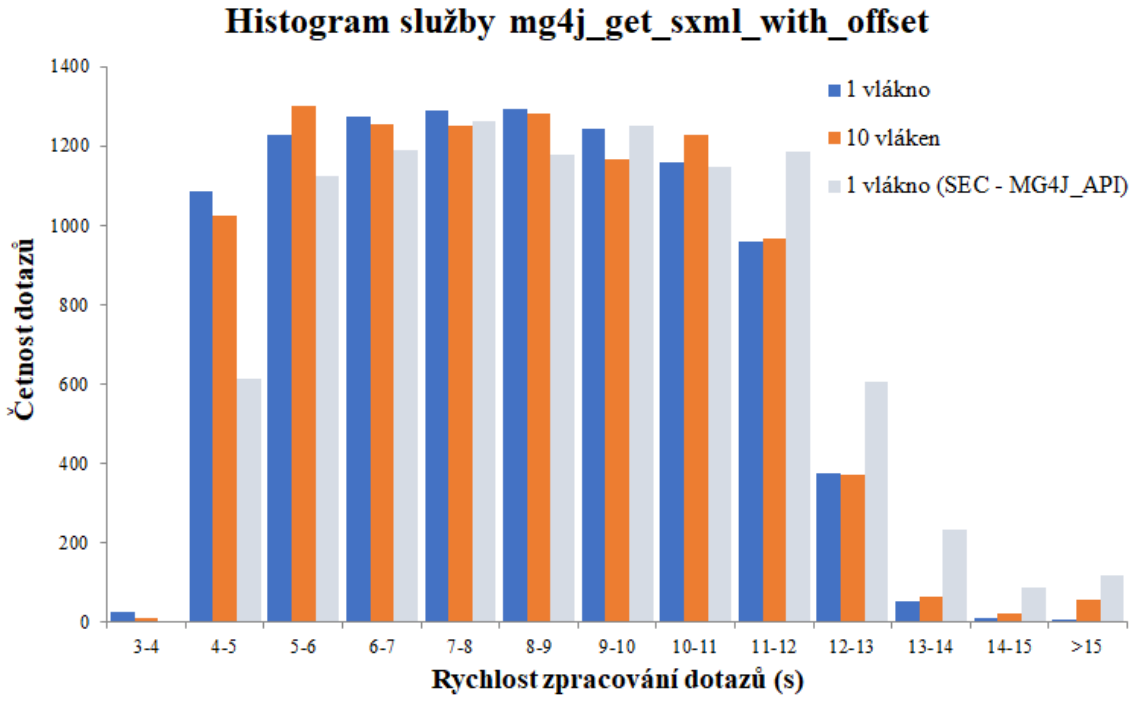
Stejně jako v histogramu 7.1 došlo k nárůstu četnosti u doby zpracování **15 a více sekund** mezi výsledky testů pro 1 dotaz a 10 dotazů naráz. Také je výrazně patrné, že četnost vyšší doby zpracování při využití MG4J_API je vyšší.

Pro přehlednější zobrazení rozdílu rychlostí mezi otestovanými službami jsem vytvořil následující jednoduchou tabulku, která obsahuje základní statistické informace. Hodnoty doby zpracování jsou platné pro **10 000 dotazů** u každého testu.

Testovaná služba	**Průměr** [s]	**Max** [s]	**Min** [s]	**Medián** [s]
get_plain_text_with_offset (1 vlákno)	7,36	31,95	2,95	7,28
get_plain_text_with_offset (10 vláken)	7,15	15,62	2,84	7,11
mg4j_get_sxml_with_offset (1 vlákno)	7,13	44,73	2,87	7,08
mg4j_get_sxml_with_offset (10 vláken)	7,21	23,34	2,86	7,12
MG4J_API (1 vlákno)	7,79	60,13	2,99	7,68

Tabulka 7.1: Statisticky zpracované výsledky testování služeb

Výsledky testování výše zmíněných služeb prokázaly, že hlavní server i vedlejší servery jsou konkurentní a jejich rychlost zpracování dotazů není výrazněji ovlivněna zpracováváním větším počtem dotazů, které byly odeslány na server naráz. Z výsledků také vyplývá, že



Obrázek 7.2: Histogram četnosti dotazů v závislosti na rychlosti zpracování služby `mg4j_get_sxml_with_offset`

průměrně uživatel čeká na zobrazení textu a anotací okolo **15 sekund**. U tohoto údaje si je třeba uvědomit velikost testovaného korpusu. S rostoucí velikostí korpusu roste i čas získaní textu a anotací.

Zajímavé jsou hodnoty maximální doby zpracování dotazu, které byly naměřeny. Z histogramů lze snadno vyčíst, že extrémní časy zpracování jsou ojedinělé. Důvodem těchto extrémů je kombinace zatíženosti školních serverů, pozice URI v souboru obsahujícím seznam všech URI vedlejšího serveru a pozice dokumentu v souboru MG4J.

7.2 Kontrola inkrementálních souborů

Testování aktualizačních služeb jsem provedl poloautomatickým způsobem. Automatické testování je velice komplikované, proto jsem od něj upustil. Poloautomaticky jsem otestoval služby `/update_sxml` a `/update_sxml_version`.

Nejdříve jsem si uložil **100 anotačních dokumentů ve formátu SXML** pomocí služby pro získání anotací v SXML. Poté jsem provedl jednoduchou změnu v SXML, kde jsem ručně pozměnil hodnotu náhodného atributu na hodnotu „*TEST_BP_2017*“. Poté jsem sekvenčně odeslal všech **100 dokumentů** na hlavní server s požadavkem o aktualizaci.

První testovanou službou byla `/update_sxml`. Po skončení všech aktualizací jsem sledoval obsah inkrementálních souborů a kontroloval jsem, zda jsou v pořádku a zda se v nich nachází řetězec „*TEST_BP_2017*“. Všechny inkrementální soubory byly v pořádku. Výsledný průměrný čas aktualizace byl **11,35 sekund**.

Druhá služba `/update_sxml_version` byla otestována stejným způsobem. Jediným rozdílem byla přidaná kontrola verzovacích souborů, která byla úspěšná, stejně jako kontrola inkrementálních souborů. Průměrný čas aktualizace byl **11,29 sekund**.

Služba `/update_sxml_batch` není v tuto chvíli podporována v MG4J GUI, které navíc momentálně nepracuje spolehlivě. Proto je velice obtížné a časově náročné vytvořit větší testovací množinu. Z tohoto důvodu jsem službu otestoval manuálně, a to pouze pro několik případů, abych otestoval její základní funkčnost.

7.3 Spuštění aktualizace znalostní databáze

Skript aktualizující KB jsem spustil na školním serveru athena5 (Intel Xeon E5-2630v2 6/12 jader, 15 MB cache, 128 GB RAM, 15 HDD v RAID6 s SSD cache).

Celkově skript nalezl **13 300 záznamů** o chybách (popsané v podkapitole 6.6 v datech od Dávida Prexty). Tyto záznamy skript vyfiltroval a zůstalo **5 094 záznamů**. Skript provedl úspěšnou aktualizaci **4 634 entit**. Pro zbylých **460 záznamů** se nepovedlo aktualizovat příslušné entity. Při manuální kontrole těchto záznamů jsem zjistil, že se jednalo o záznamy, které byly duplicitní, ale skript tyto duplicity nezachytil.

Celková aktualizace KB trvala **102 m 41 s** reálného času. Tento údaj byl získán pomocí nástroje `time`.

Aktualizovaná verze KB není v tuto chvíli veřejně dostupná. Pro pracovníky výzkumné skupiny KNOT je dostupná ve složce `/mnt/data/xvrsas00_a5/KB` na serveru athena5 s příponou `.updated`.

7.4 Známé problémy

Mezi hlavní nevyřešený problém patří posílání dotazů na URI, které obsahují na konci znak „*?*“ (například URI „*https://en.wikipedia.org/wiki/Why_Was_I_Born?*“). Knihovna requests za tímto znakem očekává parametr a jeho hodnotu. Pokud se za znakem „*?*“ nic nenachází, tak knihovna reuqests automaticky odmaže tento znak z URI, což vede k nenalezení URI na serveru a proto se pro tento typ URI neprovede žádná služba.

Mezi další problém lze zařadit značky HTML v souborech MG4J. Pro příklad uvedu značku HTML pro nezlomitelnou mezeru „` `“. Skript, který zpracovává MG4J, tyto značky nefiltruje. Problém spočívá v tom, že MG4J GUI tyto značky z textu dokumentu odstraňuje, ale neupravuje offsety anotací. To vede k tomu, že se anotace nezobrazí správně, protože jejich offsety počítají i se značkami HTML. Nutno podotknout, že je velice ojedinělé množství souborů MG4J, které obsahují značky HTML.

Kapitola 8

Závěr

V práci byly analyzovány dostupné nástroje pro práci s korpusy a to indexační nástroj MG4J, nástroj SEC a nástroj 4A.

Vytvořené řešení umožňuje získat konkretní data z rozsáhlého korpusu v uživatelsky uspokojivém čase. Další funkcí systému je zpětná aktualizace korpusu a následná aktualizace jeho indexu.

Výsledky experimentů a testování potvrdily, že je systém schopný zobrazit a aktualizovat dokumenty v korpusech během desítek sekund. Automatické testy potvrdily, že servery, které poskytují služby, jsou konkurentní. Tento fakt potvrzuje schopnost systému pracovat s požadavky více uživatelů současně.

Určitým rozšířením řešení je skript, který aktualizuje entity ve znalostní databázi. Tento skript vychází z dat, která obsahují chyby pro jednotlivé entity a jejich řešení.

V nejbližší době bude systém spuštěn na ostatních korpusech skupiny KNOT. V práci pak lze pokračovat mnoha způsoby. Systém podává rychlostně uspokojivé výsledky, ale ojediněle se objevují časově extrémní případy zobrazení dat. Tomu lze předejít například zlepšením tvorby serverových dokumentů, ve kterých jsou obsaženy URI, optimalizací vyhledávání v konkrétním souboru MG4J apod.

Dále by bylo vhodné zpracovávat a ukládat nově vytvořené anotace (resp. nové typy anotací), které nejsou kompatibilní s aktuálním formátem MG4J. V současném řešení jsou takové anotace zahozeny a jsou podporovány pouze typy anotací dostupné v MG4J. Samostatnou otázkou zůstává postup odhalení chybných anotací v korpusech. Tudíž bych se rád zabýval i nástrojem, který bude odhalovat chyby v anotacích v korpusech a navrhovat jejich řešení.

Literatura

[1] Ascher, D.: *Naučte se Python – pohotová příručka*. Grada Publishing, 2003, ISBN 9788024703671.
URL https://books.google.cz/books?id=L6Ckq7VFdDAC

[2] Balíková, M.: *indexace*. In: KTD: Česká terminologická databáze knihovnictví a informační vědy (TDKIV). Praha : Národní knihovna ČR, 2003, [Online; navštíveno 6.04.2017].
URL
http://aleph.nkp.cz/F/?func=direct&doc_number=000001549&local_base=KTD

[3] Belshe, M.; Peon, R.; Thomson, M.: *Hypertext Transfer Protocol Version 2 (HTTP/2)*. 2015, [Online; navštíveno 28.04.2017].
URL https://tools.ietf.org/html/rfc7540

[4] Boldi, P.; Vigna, S.: *MG4J at TREC 2006*. 2006, [Online; navštíveno 5.04.2017].
URL http://trec.nist.gov/pubs/trec15/papers/umilano.tera.final.pdf

[5] Boldi, P.; Vigna, S.: *MG4J (big): The Manual*. 2015, [Online; navštíveno 7.04.2017].
URL http://mg4j.di.unimi.it/man-big/manual.pdf

[6] Bray, T.: *The JavaScript Object Notation (JSON) Data Interchange Format*. 2014, [Online; navštíveno 09.05.2017].
URL https://tools.ietf.org/html/rfc7159

[7] Bray, T.; Paoli, J.; Sperberg-McQueen, C. M.; aj.: *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. 2008, [Online; navštíveno 18.04.2017].
URL https://www.w3.org/TR/REC-xml/

[8] Crockford, D.: *Introducing JSON*. 2009, [Online; navštíveno 09.05.2017].
URL http://www.json.org/

[9] Cvrček, V.; Richterová, O.: *pojmy:tag – Příručka ČNK*. 2014, [Online; navštíveno 27.04.2017].
URL http://wiki.korpus.cz/doku.php?id=pojmy:tag&rev=1416830549

[10] Doležal, J.: *Semantic Enrichment Component*. 2015, [Online; navštíveno 9.04.2017].
URL http://sec.fit.vutbr.cz/sec_documentation_cs.html

[11] Dytrych, J.: *4A Architecture*. [Online; navštíveno 9.05.2017].
URL http://knot.fit.vutbr.cz/annotations/Architecture2014.png

[12] Grešová, K.: *Corpora Processing Software*. 2016, [Online; navštíveno 18.04.2017].
URL
http://knot.fit.vutbr.cz/corpproc/corpproc_cs.html#8._b.29_Webov.C3.A9_GUI

[13] zpracování přirozeného jazyka, C.: *Úvod do korpusové lingvistiky*. 2012, [Online; navštíveno 31.03.2017].
URL https://nlp.fi.muni.cz/cs/UvodDoKorpusoveLingvistiky

[14] Lampa, P.: *Protokol HTTP 1.1*. 2002, [Online; navštíveno 16.04.2017].
URL http://www.fit.vutbr.cz/~lampa/WWW/http11.html.cs

[15] Nesselhauf, N.: *Corpus Linguistics: A Practical Introduction*. 2005, [Online; navštíveno 4.04.2017].
URL http://www.as.uni-heidelberg.de/personen/Nesselhauf/files/Corpus%20Linguistics%20Practical%20Introduction.pdf

[16] Otrusina, L.: *Knowledge Base (iotrusina)*. 2013, [Online; navštíveno 10.04.2017].
URL http://knot.fit.vutbr.cz/Decipher_NER/Decipher_NER-cs.html#Knowledge_Base_.28iotrusina.29

[17] Smrz, P.; Dytrych, J.: Advanced Features of Collaborative Semantic Annotators — The 4A System. 2015, [Online; navštíveno 02.05.2017].
URL https://www.aaai.org/ocs/index.php/FLAIRS/FLAIRS15/paper/view/10356

[18] Vidová, B. H.; Hajič, J.; Hana, J.; aj.: *Czech Academic Corpus 2.0*. Philadelphia: Linguistic Data Consortium, 2008, ISBN 1585634913.

[19] *WARC, Web ARChive file format*. 2009, [Online; navštíveno 27.04.2017].
URL https://www.loc.gov/preservation/digital/formats/fdd/fdd000236.shtml

Přílohy

Příloha A

Formát anotací v MG4J

Typ anotace	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**
osoba	jméno	pohlaví	místo narození	datum narození	místo úmrtí	datum úmrtí	zaměstnání	národnost
umělec	jméno	pohlaví	místo narození	datum narození	místo úmrtí	datum úmrtí	umělecké zaměření	národnost
umělecké dílo	název	forma	datum začátku	datum dokončení	umělecký směr	žánr	autor	-
interval	od roku	od měsíce	od dne	do roku	do měsíce	do dne	-	-
museum	název	typ	založeno	ředitel	místo	-	-	-
událost	název	začátek	konec	místo	-	-	-	-
rodina	jméno	zaměření	národnost	členové	-	-	-	-
skupina	název	zaměření	národnost	-	-	-	-	-
datum	rok	měsíc	den	-	-	-	-	-
místo	název	země	-	-	-	-	-	-
národnost	název	země	-	-	-	-	-	-
forma	název	-	-	-	-	-	-	-
prostředek	název	-	-	-	-	-	-	-
mythologie	název	-	-	-	-	-	-	-
hnutí	název	-	-	-	-	-	-	-
žánr	název	-	-	-	-	-	-	-

Tabulka A.1: Význam anotačních sloupců v MG4J